GHENT UNIVERSITY

Modelling single cell dynamics with trajectories and gene regulatory networks

Robrecht Cannoodt

Thesis submitted in fulfilment of the requirements for the degree of
Doctor in Computer Science, 2019

Supervisors:
Prof. Dr. Yvan Saeys
Prof. Dr. Katleen De Preter

FACULTY OF SCIENCES
VIB-UGENT CENTER FOR INFLAMMATION RESEARCH
DaMBi
CMGG
fwo

Vakgroep Toegepaste Wiskunde, Informatica, en Statistiek
Faculteit Wetenschappen, Universiteit Gent
Krijgslaan 281 - S2, 9000 Gent

Acknowledgements

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Contents

Nomenclature

CART	Classification And Regression Trees
DNA	Deoxyribonucleic Acid
GRN	Gene Regulatory Network
HCA	Human Cell Atlas
IM	Importance Measure
ML	Machine Learning
mRNA	Messenger RNA
NI	Network Inference
RF	Random Forests
RNA	Ribonucleic Acid
TF	Transcription Factor

CHAPTER 1

Introduction

Abstract:

Partially adapted from:
Cannoodt, R.*, Saelens, W.*, and Saeys, Y. Computational methods for trajectory inference from single-cell transcriptomics. *European Journal of Immunology* 46, 11 (2016), 2496–2506. doi:10.1002/eji.201646347.
Saelens, W.*, **Cannoodt, R.***, Todorov, H., and Saeys, Y. A comparison of single-cell trajectory inference methods. *Nature Biotechnology* 37, 5 (2019), 547–554. doi:10.1038/s41587-019-0071-9.
Todorov, H., **Cannoodt, R.**, Saelens, W., and Saeys, Y. Network Inference from Single-Cell Transcriptomic Data. *Gene Regulatory Networks* (2019), 235–249. doi:10.1007/978-1-4939-8882-2_10.
* Equal contribution

1

1.1 The cell

The cell is the smallest unit of life, of which all known living organisms are composed. Every cell houses a plethora of biomolecular processes that allow it to adapt to changes in their environment continuously. It can be very challenging to comprehend the cellular response to a signal due to the dynamic nature of these processes. A reductionist approach to understanding a complex biological system is to study the biochemical components which it is comprised of [1].

Reductionist biologists are delighted by recent advances in experimental technologies that permit measuring the abundance of thousands of different biochemical molecules in tens of thousands of individual cells. Observing the biomolecular insides of cells in this manner will ultimately provide fundamental insights into the processes that govern these cells and help uncover novel approaches for diagnosing and treating disease. Every coin has its flip side, however, and in this case, it is that the amount of data generated from these experiments is not analysable by hand.

For example, the Human Cell Atlas (HCA) consortium [2] has set out to develop a comprehensive reference map of all the different types of cells in the human body. Experts in the field often metaphorically describe the HCA initiative as aiming to develop a 'Google Maps' of the human body. Even in its infancy, the HCA has profiled 3.8 million cells from 248 donors across 42 labs [3], and this number is likely to increase well above one hundred million.

The sheer volume of the data generated from such highly-integrative and high-throughput experiments are not the only reason why they are so challenging to interpret. Namely, the data inherently also suffers from batch effects arising from differences between donors and labs, and also contains high levels of noise arising from the experimental profiling techniques used [4]. Biologists thus turn to computer scientists[1] to develop new tools to tackle these problems and help biologists extract meaningful biological insights from the data.

This work makes incremental contributions to the field in order to be able to address the aforementioned problems in a more comprehensive context. This chapter first introduces several key concepts in both cell biology and computer science, upon which the remainder of this work relies. Afterwards, the research objectives and main contributions of this work are outlined.

1.1.1 The origins of life and the RNA world

The discovery of the double helix shape of Deoxyribonucleic Acid (DNA) [5] is often considered the pivot point in our understanding of the origins of life and evolution. A modern high-school student can plausibly be expected to know that DNA serves as a medium for storing the genetic information required to reproduce a whole organism. With other words, the DNA of an organism contains the complete set of instructions required to build all of the biomolecular machinery present in its body. The magnitude of this discovery is reflected in our language and culture alike; with sayings such as "It's in your DNA.", or usage of its shape in countless illustrations or artworks (Figure 1.1).

Even so, a widely-accepted hypothesis states that life (or cells) did not originate from DNA, but instead was kicked off from its lesser-known cousin, Ribonucleic Acid (RNA). According to the RNA world hypothesis [6], the very first primitive cells used RNA both to store genetic information and perform the chemical reactions required to sustain themselves (Figure 1.2). Only later did cells develop the ability to use DNA and proteins to self-sustain in a process commonly referred to as the Central Dogma.

[1] or computational biologists turn to themselves

Figure 1.1: A prominent display of the double helix shape at the VIB FSVM building.

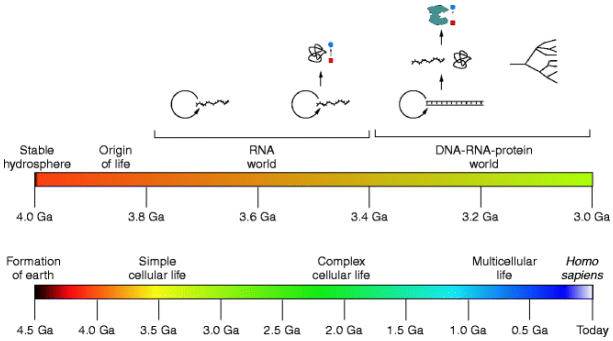


Figure 1.2: RNA world. The postulated rise and fall of the RNA world during the evolution of life, from early self-replicating RNA to complex, RNA-controlled metabolism, to the invention of translation, followed by diversification of all modern branches of life. Image from Horning (2011) [7].

1.1.2 The Central Dogma

The Central Dogma is a set of processes present which govern the general flow of genetic information in almost all existant living cells. In short, it states that DNA codes for RNA, which in turn codes for proteins. In this work, we assume the main processes involved in the Central Dogma are replication, transcription, splicing and translation (Figure 3).

Replication is the process of duplicating DNA, which allows a cell to divide such that the resulting daughter cells retain complete copies of its genetic information. DNA consists of four different so-called nucleobases named adenine (A), cytosine (C), guanine (G), and thymine (T). Each strand of the double helix structure of DNA is a linear chain of nucleobases. The two strands are held together by hydrogen bonds since adenine can form three hydrogen bonds with thymine and cytosine can form two hydrogen bonds with guanine. Since each nucleobase can only bond with one other nucleobase, one strand is complementary to the other. Thus a strand of DNA can be represented by a sequence of letters, for example, "ACTCGGTTTAGCA".

A stretch of DNA that contains a genetic blueprint for a particular molecule is called a gene, and the collection of an organism's genes is called its genome. **Transcription** is the process of synthesising an RNA molecule from a gene, and the resulting molecule is called a transcript. RNA is similar to DNA but differs in several ways; most notably all thymine nucleobases are replaced with uracil (U), and RNA molecules consist of only one strand. Due to its single-strandedness, RNA is less stable and will break down faster. Single-strandedness also allows some types of RNA (e.g. transfer RNA, ribosomal RNA) to form more complex three-dimensional structures by having certain regions bind to other regions of the strand. This work only considers messenger RNA (mRNA). mRNAs are transcribed from protein-coding genes, meaning that the mRNAs will lead to the production of protein molecules through the process of translation.

RNA splicing is a process that occurs in almost all organisms (but not all) and results in the removal of specific regions in the RNA molecule. Introns are the regions removed by this process, and exons

the regions that remain. The main functionality of splicing is to be able to create multiple variants of the same product, which can affect the enzymatic properties or localisation of the resulting product [8].

During **translation**, a chain of amino acid is synthesised from an mRNA transcript. Every three nucleobases are translated into one of 21 different amino acids. The resulting chain of amino acids is folded up into a protein, the structure of which is determined by the sequence of different amino acids in the chain. In turn, its structure determines the functionality of the protein, which includes catalysing biochemical reactions, providing structure, and transportation of molecules.

1.1.3 Cell types

The functionality provided by a cell is defined (mostly) by the proteins of which it consists. One common approach to trying to understand the functionality of a cell is to observe which molecules are present in the cell and to associate those molecules with functionality.

Homo sapiens like to categorise everything they encounter and so too have they conceptualised groups of cells called "cell types" according to their functionality. The concept of cell types eases reasoning about all aspects of biology, for instance, which cell types turn into (differentiate) or communicate with which other cell types, or how a cell type responds to a specific stimulation. Cells can be highly specialised toward performing a particular function (e.g. memory B cells accelerate immune response by remembering previously encountered pathogens), while other cells maintain a strong ability to differentiate into other cell types.

Cell differentiation is not an instant process; it is a continuous process in which a cell gradually produces the biochemical machinery required in order to fulfil a particular task. In this regard, it makes sense not only to reason about cell types but also about the transition states between cell types and the dynamic processes involved therein.

1.1.4 Cell dynamics and gene regulation

If cells are dynamic entities and can gradually produce the molecules needed to acquire new functionality, what is the process by which this happens? The mechanism by which this happens is called gene regulation. Some proteins (or other molecules such as micro RNAs) are capable of determining the rate at which a gene is transcribed (transcription rate). Such proteins are called transcription factors (TFs), and the genes it regulates are called its targets. Typically, one TF will regulate the transcription rate of many targets.

Production of a specific molecule might require multiple cascades of gene regulation. The collection of all gene regulatory interactions between transcription factors and targets is called a gene regulatory network (GRN). Studying the active parts of a cell's gene regulatory network can thus reveal which dynamic processes are taking place.

1.1.5 Profiling single cells

In order to understand a biological process, it is often quite helpful to be able to profile (i.e. observe) the biomolecular components involved therein. The single-cell "omics" technologies which we have at our fingertips today originated from the convergence of two different fields, *"single-cell"* and *"omics"*.

The earliest approaches for measuring the abundance of particular molecules in *single cells* used the preferred instrument of every stereotypical biologist: the microscope. Since it was developed

by Coons et al. in 1941, immunohistochemistry (IHC) has been instrumental in visualising antigen-antibody proteins [9]. In many multicellular organisms, antibodies and antigens serve as crucial communication tools as part of the organism's immune system. A cell can present a particular type of antigen on its cell surface, which allows a particular type of antibody to bind to it.

IHC (and many other biotechnologies) visualises antigen-antibody reactions by attaching particular molecules to the antibody, such as an enzyme that catalyses a colour-producing reaction, or a fluorescent chemical compound that can re-emit light upon light excitation. Using different colours (wavelengths) allows measuring expression levels of different antibodies simultaneously. Characterising cells in a quantifiable way is labour intensive; however, since it involves acquiring an image of many cells and drawing a contour around each cell (called cell segmentation). While modern implementations of IHC improve the throughput drastically by using robots to automate the image acquisition and computer software to automate cell segmentation, the procedure is still labour intensive as the robots and computer software still needs to be kept in check.

Flow cytometry [10] is a technique which circumvents imaging and segmentation issues by having a steady stream of cells run through a laser and measuring the amount of light scattered from those cells. Flow cytometry technology enables to measure protein expression levels for millions of cells and tens of different antibodies.

Since IHC and flow cytometry, many new technologies have been developed which allow quantifying expression levels of molecules in single cells (e.g. mass cytometry, single-cell qPCR, FISH). All of these single-cell (non-omics) technologies are limited by the number of different molecules they could measure, however; and thus required handpicking the molecules of interest before performing an experiment, making the experiment biased towards the preconceptions of the experimenter.

On the other side of the spectrum are the so-called "omics" technologies. "Omics"[2] is a collective term for profiling all molecules of a particular type in a high-throughput manner. There are many types of "omics", but the most commonly used are the following. In genomics, all of an organism's genes are studied – its whole genome. Transcriptomics and proteomics study the organisms RNA transcripts and proteins, respectively. A notable downside of traditional omics technologies is that in order to capture enough material an ensemble of cells needs to be profiled, and thus only the average expression levels are returned; thereby granting the technology the name "bulk" omics. If a subset of these cells contains unique patterns in expression levels, this pattern will be masked in the bulk population and is thus undetectable. Specific examples of omics technologies are next-generation sequencing, which can be used to determine the DNA sequence of an organism, and RNA sequencing, which profiles the sequences of RNA transcripts. By mapping the sequences of RNA transcripts to genes in the organisms DNA, a gene expression profile can be obtained.

Transformative technological advances in microvolume sequencing allowed Tang et al. to analyse the transcriptome at single-cell resolution [12], thereby bringing single-cell biology and omics together to create single-cell omics (Figure 1.3A). During the decade that followed, the number of single-cell omics technologies has skyrocketed, allowing to profile tens of thousands of cells (Figure 1.3B) and measuring other levels of information such as proteomic expression levels (Figure 1.3C).

The rapidly advancing field of single-cell omics harbours exceptional opportunities to discover new aspects of biology and redefine existing knowledge. Some of these opportunities lie in efforts like the Human Cell Atlas. The HCA consortium has set out to redefine all human cell types in terms of their gene expression and location, and the developmental trajectories connecting the different cell types. As part of this endeavour, the consortium will likely profile the whole transcriptomes tens or even hundreds of millions of cells.

[2] The etymology of "omics" is quite interesting [11].

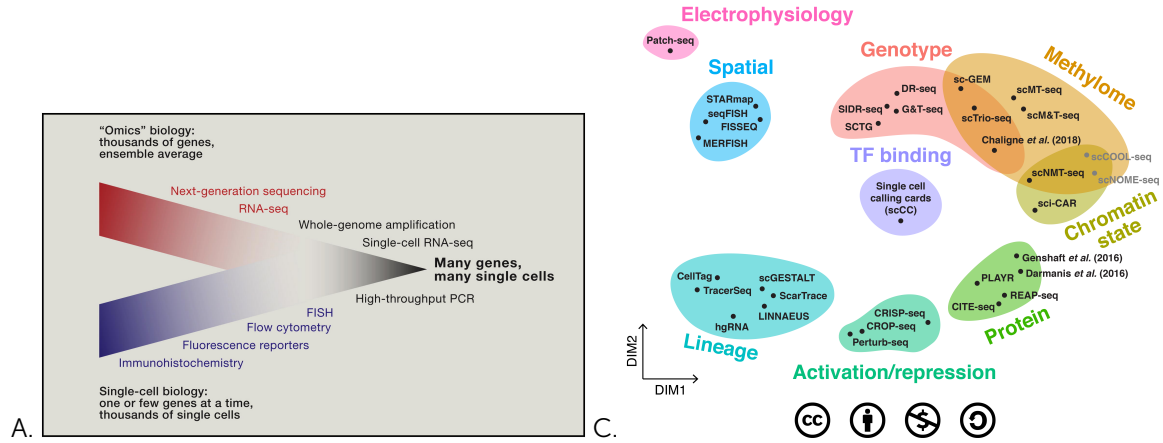


Figure 1.3: A. **Convergence of "Omics" Biology and Single-Cell Biology.** Technology that allows researchers to obtain genome-wide information from single cells is extending the boundaries of a field that has thus far been limited to the analyses of a select gene in eukaryotes. Image from Junker and van Oudenaarden (2014) [13]. B. C. scmultiomics [14].

1.2 Computational tools

The rapidly advancing field of single-cell omics harbours exceptional opportunities to discover new aspects of biology and redefine existing knowledge.

Some of these opportunities lie in efforts such as the Human Cell Atlas. The HCA consortium has set out to redefine all human cell types in terms of their gene expression and location, and the developmental trajectories which connect the different cell types. As part of this endeavour, the consortium will perform single-cell omics on tens or even hundreds of millions of cells.

Single-cell omics permits new types of analyse but also come with hitherto unseen data characteristics, the combination of which poses exciting new challenges for the computational community to tackle (Figure 1.4A)[15, 16, 17]. These challenges include:

- normalisation: separating biological noise from technical noise,
- dimensionality reduction: providing a visual and informative overview of a given dataset,
- trajectory inference: identifying and characterising transitions between different cellular states, and
- gene regulatory network inference: inferring regulatory interactions between transcription factors across individual cells.

1.2.1 Normalisation

1.2.2 Dimensionality reduction

Single-cell omics datasets typically have too many dimensions (features) in order to be easily interpretable by humans and even by most computational tools. Dimensionality reduction (DR) methods transform high-dimensional data into a meaningful representation with fewer dimensions. It is important to note that its usage depends on the target audience: for humans – to visualise data in a 2-D plane to aid with interpretation by humans, or for computers – to construct a denser representation of the data such that it mostly contains the same information but with fewer dimensions.

There are many ways of classifying DR methods [20], but this work will use the following main categories: feature projection-based and manifold learning. Projection-based DR methods aim to

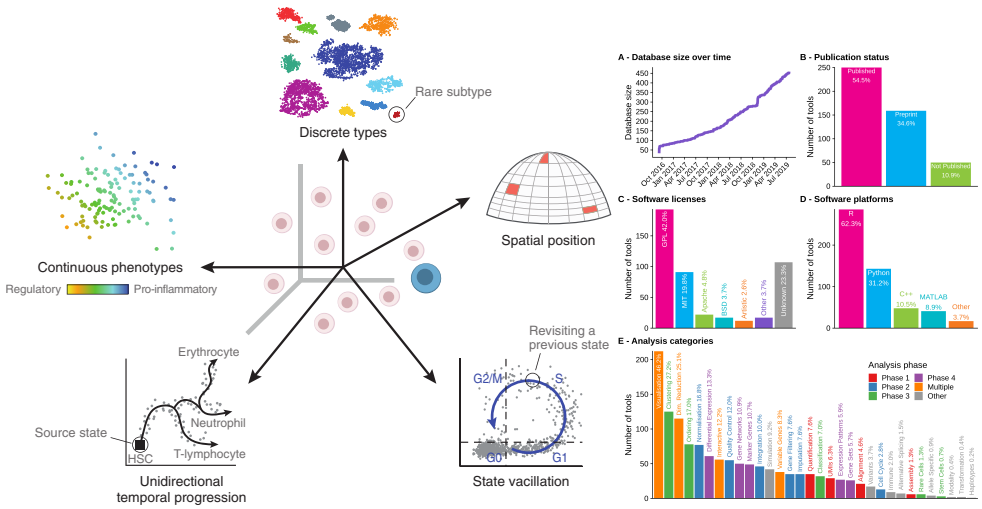


Figure 1.4: A. Single-cell omics allows for many new types of computational approaches. Figure adapted from Wagner et al. (2016) [18]. B. Zappia et al. (2018) [19]

perform a linear transformation of the data while preserving the pairwise distances between samples as much as possible. Examples of commonly used projection-based DR methods in single-cell omics are PCA and MDS. Manifold learning methods are methods which reconstruct a higher-order structure in the original space (e.g. a graph or a grid), visualising the structure in a lower-dimensional space, and mapping the original samples to the lower-dimensional space. Manifold learning can be an iterative optimisation process using a predefined criterion. Examples of manifold learning techniques are t-SNE, Diffusion Maps and UMAP.

1.2.3 Trajectory inference

Single-cell omics data provide new opportunities for studying cellular dynamic processes, such as the cell cycle, cell differentiation and cell activation [21, 22]. Trajectory inference (TI) is a new category of computational tools used to offer an unbiased and transcriptome-wide understanding of a dynamic process [21, 23].

The dataset can be a single snapshot of a mixture of cells in different stages, or a set of samples collected at different time points (Figure 1.5A). Typically, TI methods first analyse similarities between cells, optionally infer the topology of the underlying process, and finally order cells along that trajectory (Figure 1.5B). The second step can be optional, as some methods assume a specific topology beforehand. TI methods allow the identification of new subsets of cells, delineation of a differentiation tree, and characterisation of the main driver genes along a state transition (Figure 1.5C). Current applications of TI focus on specific subsets of cells, but ongoing efforts to construct transcriptomic catalogs of whole organisms [24, 25, 26] underline the urgency for accurate, scalable [27, 28] and user-friendly TI methods.

1.2.4 Gene regulatory network inference

Gene regulatory network inference, or network inference (NI) for short, is a type of computational analysis where thousands of transcriptomic profiles are analysed together in order to infer the regulatory interactions between transcription factors and genes. This topic already received much attention with the advent of bulk omics (before single-cell omics). These efforts culminated in several DREAM competitions assessing the performance of 29 different NI methods [29, 30].

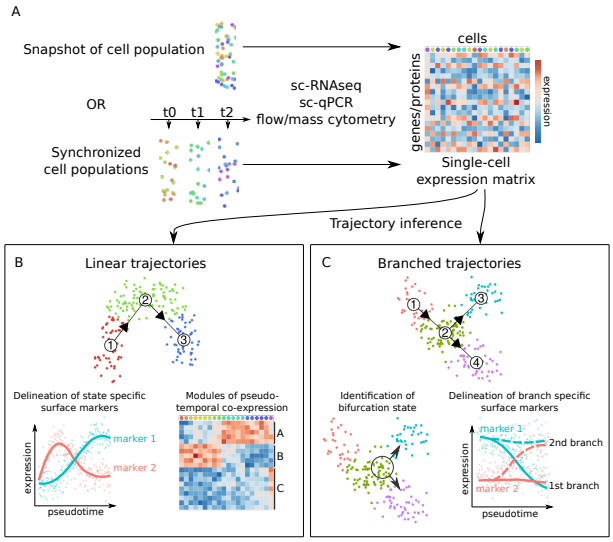


Figure 1.5: Applications of single-cell trajectory inference methods. (A) Single-cell omics data appropriate for TI can be both obtained from an unsynchronised population of single cells (snapshot data) but also from synchronised cell populations. (B) UPDATE! (C) UPDATE!

After the last DREAM competition, it seemed that interest in NI methodology had declined. After all, NI on bulk omics profiles suffered from several crucial issues. As mentioned previously, bulk profiles are generated by pooling together the RNA transcripts of a supposedly homogeneous population of thousands of cells. Since the expression values are averaged over the whole population, incorrect assumptions on the homogeneity of the pooled cells may lead to the masking of relevant expression patterns in rare cell populations (Figure 1.6). Besides, NI methods rely on a diverse set of time-series and perturbation experiments in order to reliably identify causal regulatory interactions. Such experiments are expensive and time-consuming, and an inaccurate selection of time points might result in crucial intermediate stages being missed.

The advent of single-cell omics has made scientists wonder whether now is the time to revisit network inference [15]. One of the main advantages of single-cell omics is the ability to quantify the exact cellular state of thousands of cells per experiment. The heterogeneity between cells caused by naturally occurring biological randomness [31] can be exploited to infer regulatory interactions between TFs and their target genes at much lower costs (see Figure 1.6). In this setting, heterogeneity in the cell population eases network inference, rather than mask condition-specific expression patterns in regulatory interactions.

1.3 Research objectives

Recent technological advancements in profiling single cells are having significant repercussions in many fields of biology. The new types of analyses made possible warrant the development of computational tools which can solve the problem at hand and deal with the new data characteristics, ultimately in order to create useful and accurate hypotheses for biologists.

In response, scientists all over the globe have been working hard to tackle these computational challenges, resulting in more than 470 software tools for processing single-cell omics data in one form or another [19]. More than half of these tools were published in peer-reviewed journals.

The high number of tools available presents the user choice, but also uncertainty and indecision about which tool to use for the task at hand. Choosing the method with most citations might result in a weaker result than picking a method at random, and making an informed decision requires spending

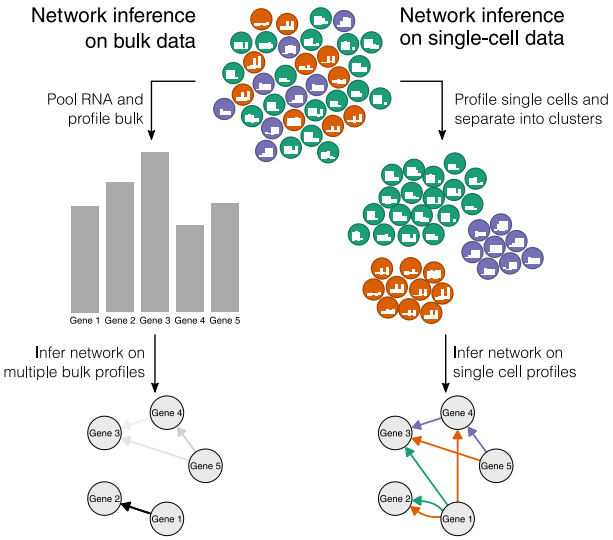


Figure 1.6: Bulk expression data return the average expressions of genes among large numbers of cells. In order to infer regulatory networks from this type of data, multiple bulk profiles (resulting from time series or perturbation experiments) are required. On the other hand, sequencing the transcriptome at the single-cell level uncovers the high variability among cells, providing the necessary information to infer gene regulatory networks directly.

an enormous amount of time in comparing the results of numerous methods, on various platforms, on multiple different datasets.

The field of single-cell omics needs quantitative ways to measure the performance of different categories of tools. Clear quantitative metrics allow performing comprehensive studies to assess the advantages and disadvantages of existing tools. In addition, they could also serve as a minimum performance criterion before the publication of a new tool.

In the computational jungle of single-cell omics, experts act as beacons of hope by sharing guidelines based on comprehensive benchmarking studies. Their disseminations (in the form of manuscripts [32, 33], courses [34, 35], and slides shown during keynote caffeine refuelling sessions [36]) are crucial in leading new users, and ultimately the whole field, to better practices for performing single-cell omics analyses.

1.4 Outline

This work aims to tackle multiple computational problems in extracting biological hypotheses from single-cell omics data. (TODO: rephrase that we focus on datasets that contain developing cells, and that the tools we develop make this assumption.) We introduce a framework for evaluating multiple types of single-cell omics tools, more specifically dimensionality reduction, trajectory inference, and gene regulatory network inference (Chapter 2). Next, we perform a comprehensive evaluation of 45 TI methods and construct guidelines based on our observations (Chapter 3). As part of the evaluation, we developed a framework to infer, visualise, and interpret trajectories, which we made available as a separate tool called dyno (Chapter 4). We also introduce two new computational tools of our own. SCORPIUS is a TI method specialised in inferring linear trajectories (Chapter 5). bred is a true single-cell NI method which can infer gene regulatory networks for individual cells (Chapter 6).

Chapter 2 – why dyngen, what is it, what can it be used for.

Chapter 3 – why dynbenchmark, what is it, what can it be used for.

Chapter 4 – why dyno, what is it, what can it be used for.

Chapter 5 – why scorpius, what is it, what can it be used for.

Chapter 6 – why bred, what is it, what can it be used for.

1.5 List of contributions

1.5.1 First-author publications

- **Cannoodt R** *, Saelens W *, Saeys Y. Computational methods for trajectory inference from single-cell transcriptomics. European journal of immunology. 2016 Nov;46(11):2496-506.
- **Cannoodt R**, Saelens W, Sichien D, Tavernier S, Janssens S, Guilliams M, Lambrecht B, De Preter K, Saeys Y. SCORPIUS improves trajectory inference and identifies novel modules in dendritic cell development. bioRxiv. 2016 Oct:079509.
- **Cannoodt R**, Ruyssinck J, Ramon J, De Preter K, Saeys Y. IncGraph: Incremental graphlet counting for topology optimisation. PloS one. 2018 Apr 26;13(4):e0195997.
- Saelens W *, **Cannoodt R** *, Todorov H, Saeys Y. A comparison of single-cell trajectory inference methods. Nature biotechnology. 2019 May;37(5):547.
- dyngen, submitted?
- dyno, submitted?
- scorpius, submitted?
- bred, submitted?

*: Equal contribution.

1.5.2 Co-author publications

- Decock A, Ongenaert M, **Cannoodt R**, Verniers K, De Wilde B, Laureys G, Van Roy N, Berbegall AP, Bienertova-Vasku J, Bown N, Clément N. Methyl-CpG-binding domain sequencing reveals a prognostic methylation signature in neuroblastoma. Oncotarget. 2016 Jan 12;7(2):1960.
- Van Cauwenbergh C, Van Schil K, **Cannoodt R**, Bauwens M, Van Laethem T, De Jaegere S, Steyaert W, Sante T, Menten B, Leroy BP, Coppieters F. arrEYE: a customized platform for high-resolution copy number analysis of coding and noncoding regions of known and candidate retinal dystrophy genes and retinal noncoding RNAs. Genetics in Medicine. 2017 Apr;19(4):457.
- Claeys S, Denecker G, **Cannoodt R**, Kumps C, Durinck K, Speleman F, De Preter K. Early and late effects of pharmacological ALK inhibition on the neuroblastoma transcriptome. Oncotarget. 2017 Dec 5;8(63):106820.
- Depuydt P, Boeva V, Hocking TD, **Cannoodt R**, Ambros IM, Ambros PF, Asgharzadeh S, Attiyeh EF, Combaret V, Defferrari R, Fischer M. Genomic amplifications and distal 6q loss: novel markers for poor survival in high-risk neuroblastoma patients. JNCI: Journal of the National Cancer Institute. 2018 Mar 5;110(10):1084-93.
- Scott CL, T'Jonck W, ..., **Cannoodt R**, Saelens W ..., Guilliams M. The transcription factor ZEB2 is required to maintain the tissue-specific identities of macrophages. Immunity. 2018 Aug 21;49(2):312-25.

- Saelens W, **Cannoodt R**, Saeys Y. A comprehensive evaluation of module detection methods for gene expression data. Nature communications. 2018 Mar 15;9(1):1090.
- Todorov H, **Cannoodt R**, Saelens W, Saeys Y. Network Inference from Single-Cell Transcriptomic Data. InGene Regulatory Networks 2019 (pp. 235-249). Humana Press, New York, NY..
- Van den Berge K, De Bezieux HR, Street K, Saelens W, **Cannoodt R**, Saeys Y, Dudoit S, Clement L. Trajectory-based differential expression analysis for single-cell sequencing data. BioRxiv. 2019 Jan 1:623397.
- Weber LM, Saelens W, **Cannoodt R**, Soneson C, Hapfelmeier A, Gardner PP, Boulesteix AL, Saeys Y, Robinson MD. Essential guidelines for computational method benchmarking. Genome biology. 2019 Dec;20(1):125.

1.5.3 Open-source software

As part of this work, many open-source software packages were created and many others were contributed to (Table 1.1).

Packages that were created as part of this work are hosted on Github under the username rcannood[3] or the dynverse organisation[4]. As part of our standard development practices, we automate execution of unit tests and writing extensive documentation to ensure the code complies with CRAN policy before submission. We aim to submit all other packages to CRAN as well.

We also helped maintain or extend other packages on Github, CRAN or Bioconductor on which our software depends. This includes help speed up parts of the dependency (slingshot), adding new functionality (devtools, ParamHelpers), fixing bugs (proxyC, rlang, monocle, splatter, slingshot), becoming a maintainer of orphaned packages (princurve, GillespieSSA), and extending the documentation (devtools, mlr, remotes). Several of these package receive millions of downloads per year (devtools, remotes, rlang).

[3] https://github.com/rcannood?tab=repositories
[4] https://github.com/dynverse?tab=repositories

Table 1.1: Contibutions to open-source software. Following abbreviations denote the role within the package: aut Author, ctb Contributor. Yearly download statistics are based on the number of downloads between 2019-08-01 and 2019-08-23. CRAN download statistics are retrieved from the Rstudio CRAN mirror only; other CRAN mirrors do not track download statistics. For Github repositories, no download statistics could be retrieved.

Name	Role	Host	Downloads per year	Description
babelwhale	aut	CRAN	3491	Interacting with Docker and Singularity containers
dynbenchmark	aut	Github		Pipeline for benchmarking trajectory inference methods
dyndimred	aut	CRAN	571	Applying dimensionality reduction methods
dyneval	aut	Github		Evaluating trajectory inference methods
dynfeature	aut	Github		Calculating feature importance scores from trajectories
dyngen	aut	Github		Simulating single-cell data using gene regulatory networks
dynguidelines	aut	Github		User guidelines for trajectory inference
dynmethods	aut	Github		A collection of wrappers for trajectory inference methods
dyno	aut	Github		A pipeline for inferring, visualising and interpreting trajectories
dynparam	aut	CRAN	2539	Creating meta-information for parameters
dynplot	aut	Github		A simple visualisation library for trajectories
dynplot2	aut	Github		A fully customisable visualisation library for trajectories
dyntoy	aut	Github		Generating simple toy data of cellular differentiation
dynutils	aut	CRAN	4570	Common functionality for the dynverse packages
dynwrap	aut	Github		A common format for trajectories
GillespieSSA	aut	CRAN	7855	Gillespie's Stochastic Simulation Algorithm (SSA)
GillespieSSA2	aut	CRAN	2253	Gillespie's Stochastic Simulation Algorithm for Impatient People
gng	aut	Github		An Rcpp implementation of the Growing Neural Gas algorithm
incgraph	aut	CRAN	2809	Incremental graphlet counting for network optimisation
princurve	aut	CRAN	24'439	Fits a principal curve in arbitrary dimension
proxyC	aut	CRAN	112'753	Computes proximity in large sparse matrices
qsub	aut	CRAN	2698	Running commands remotely on gridengine clusters
SCORPIUS	aut	CRAN	4824	Inferring developmental chronologies from single-cell RNA sequencing data
ClusterSignificance	ctb	Bioc	571	Assess if class clusters in dimensionality reduced data representations have a separation different from permuted data
devtools	ctb	CRAN	3'254'816	Tools to make developing R packages easier
merlot	ctb	Github		A method for reconstructing lineage-tree topologies from scRNA-seq data
mlr	ctb	CRAN	141'192	Machine Learning in R
monocle	ctb	Bioc	29'121	Clustering, differential expression, and trajectory analysis for single-cell RNA-Seq
ParamHelpers	ctb	CRAN	101'137	Helpers for Parameters in Black-Box Optimization, Tuning and Machine Learning
pseudogp	ctb	Github		Probabilistic pseudotime for single-cell RNA-seq
remotes	ctb	CRAN	3'476'435	R package installation from remote repositories, including GitHub
rlang	ctb	CRAN	10'314'725	Functions for base types and core R and tidyverse features
SCope	ctb	Github		Visualization of large-scale and high dimensional single cell data
slingshot	ctb	Bioc	9538	Tools for ordering single-cell sequencing
splatter	ctb	Bioc	2999	Simple simulation of single-cell RNA sequencing data
URD	ctb	Github		URD reconstructs transcriptional trajectories underlying specification or differentiation processes in the form of a branching tree from single-cell RNAseq data
wishbone	ctb	Github		Identify bifurcating developmental trajectories from single-cell data

CHAPTER 2

dyngen: simulating single cells

Abstract:

2

2.1 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 [23]. 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 [37]. 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 [38] and PROSSTT [39]). 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 2.1). dyngen was initially developed as part of a comprehensive benchmark of TI methods [40] but has since been extended to be applicable in a much broader context. We demonstrate its flexibility by simulating numerous different types of biological experiments, and using these simulations to develop new benchmarking techniques for computational tools.

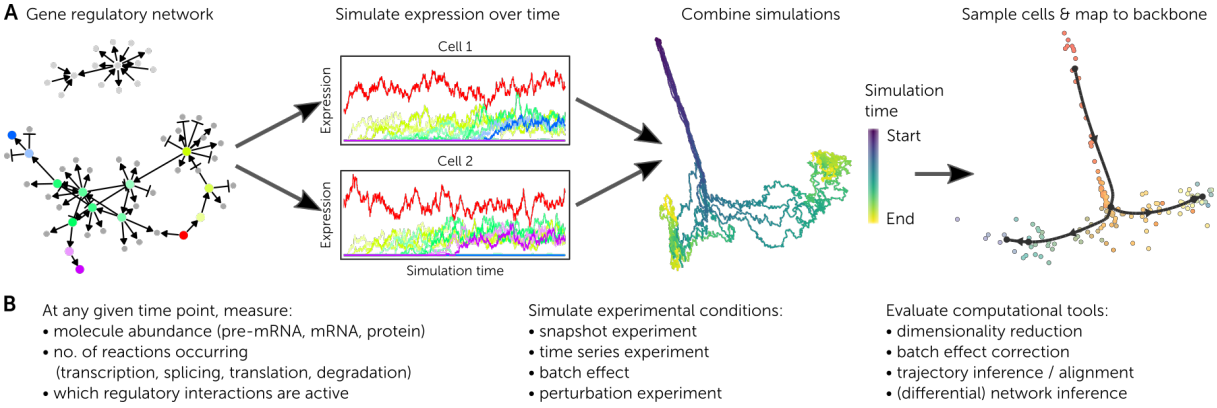


Figure 2.1: Showcase of dyngen functionality. **TODO: change to pdf.** Remove B?

2.2 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 2.2). 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 2.2A). These reactions are

determined from a predefined set of gene regulatory interactions (Figure 2.2B), 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, and regulation activations (Figure 2.2C–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.

2.3 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; several examples are given.

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

2.4 Methods

2.4.1 Simulating a snapshot experiment with dyngen

Define the module network and the state network

Generate the transcription factor network

Generate targets and housekeeping genes

Convert gene regulatory network to a set of SSA reactions

Simulate backbone

Run SSA simulations

Map SSA simulations to backbone

Simulate snapshot experiment

2.4.2 Extensions

Predefined backbones

Backbone lego

Time series experiment

Perturbation experiment

Batch effects

2.4.3 Example use cases

Trajectory alignment

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

Differential network inference

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

RNA velocity

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

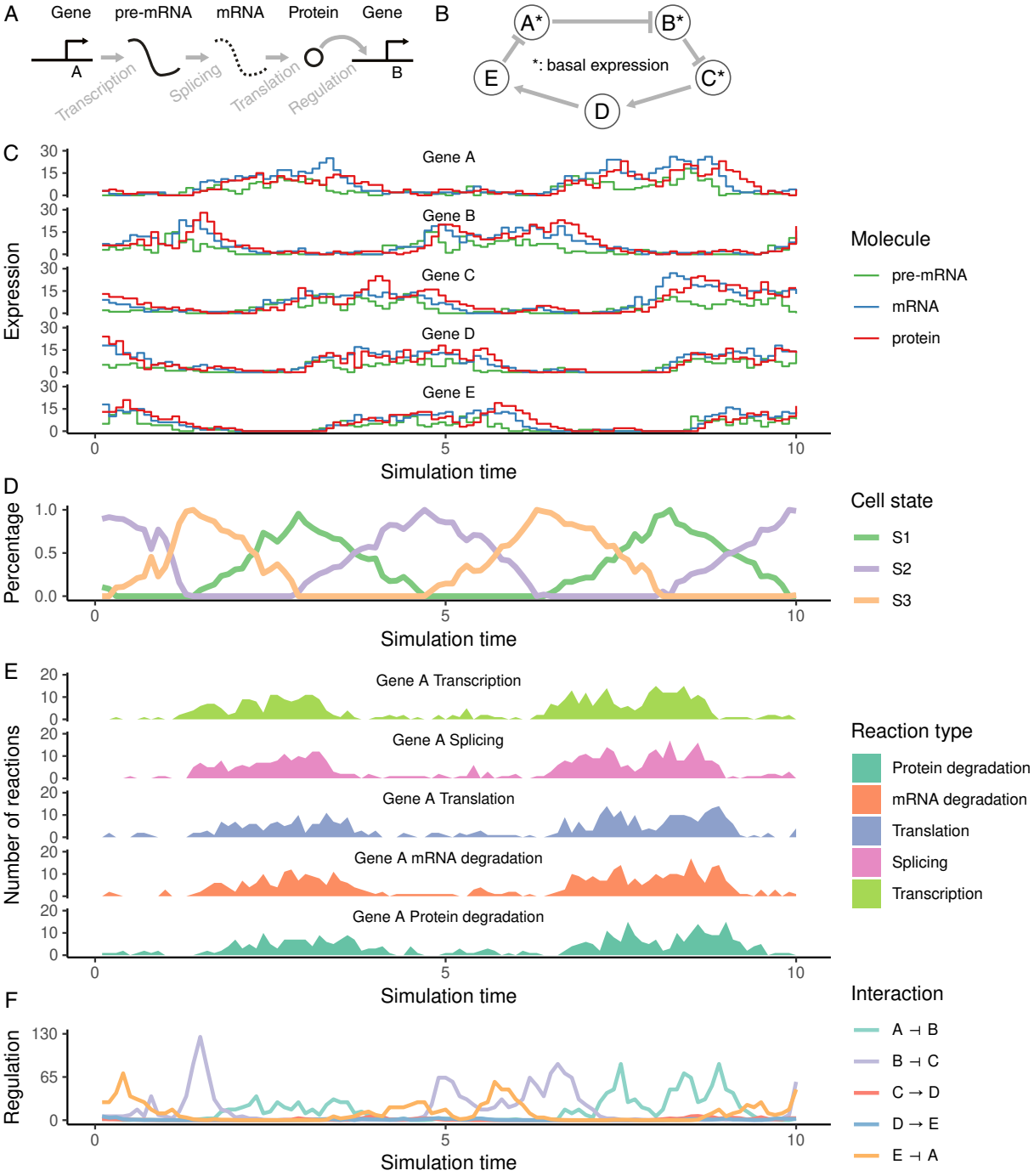


Figure 2.2: Showcase of dyngen functionality. A time resolution of 0.1 was used, but this can be increased or decreased without effect on performance of the execution of the simulation. **TODO: perhaps it's better to replace Figure 2.2 with one subfigure for each of the paragraphs in this text.**

CHAPTER 3

dynbenchmark: A comparison of single-cell trajectory inference methods

Abstract: Trajectory inference approaches analyze genome-wide omics data from thousands of single cells and computationally infer the order of these cells along developmental trajectories. Although more than 70 trajectory inference tools have already been developed, it is challenging to compare their performance because the input they require and output models they produce vary substantially. Here, we benchmark 45 of these methods on 110 real and 229 synthetic datasets for cellular ordering, topology, scalability and usability. Our results highlight the complementarity of existing tools, and that the choice of method should depend mostly on the dataset dimensions and trajectory topology. Based on these results, we develop a set of guidelines to help users select the best method for their dataset. Our freely available data and evaluation pipeline (benchmark.dynverse.org) will aid in the development of improved tools designed to analyze increasingly large and complex single-cell datasets.

Adapted from:
Saelens, W.*, **Cannoodt, R.***, Todorov, H., and Saeys, Y. A comparison of single-cell trajectory inference methods. *Nature Biotechnology* 37, 5 (2019), 547–554. doi:10.1038/s41587-019-0071-9.

* Equal contribution

3.1 Introduction

Single-cell omics data, including transcriptomics, proteomics and epigenomics data, provide new opportunities for studying cellular dynamic processes, such as the cell cycle, cell differentiation and cell activation [21, 22]. Such dynamic processes can be modeled computationally using trajectory inference (TI) methods, also called pseudotime analysis, which order cells along a trajectory based on similarities in their expression patterns [41, 23, 42]. The resulting trajectories are most often linear, bifurcating or tree-shaped, but more recent methods also identify more complex trajectory topologies, such as cyclic [43] or disconnected graphs [44]. TI methods offer an unbiased and transcriptome-wide understanding of a dynamic process[21], thereby allowing the objective identification of new (primed) subsets of cells [45], delineation of a differentiation tree [46, 47] and inference of regulatory interactions responsible for one or more bifurcations [27]. Current applications of TI focus on specific subsets of cells, but ongoing efforts to construct transcriptomic catalogs of whole organisms [24, 25, 26] underline the urgency for accurate, scalable [27, 28] and user-friendly TI methods.

A plethora of TI methods has been developed over the past few years and even more are being created every month (Supplementary Table 1). Indeed, in several repositories listing single-cell tools, such as omictools.org [48], the 'awesome-single-cell' list [49] and scRNA-tools.org [50], TI methods are one of the largest categories. While each method has its own unique set of characteristics in terms of underlying algorithm, required prior information and produced outputs, two of the most distinctive differences between TI methods are whether they fix the topology of the trajectory and what type(s) of graph topologies they can detect. Early TI methods typically fixed the topology algorithmically (for example, linear [51, 45, 52, 53] or bifurcating trajectories [54, 55]) or through parameters provided by the user [56, 57]. These methods therefore mainly focus on correctly ordering the cells along the fixed topology. More recent methods also infer the topology [58, 59, 44], which increases the difficulty of the problem at hand, but allows the unbiased identification of both the ordering inside a branch and the topology connecting these branches.

Given the diversity in TI methods, it is important to quantitatively assess their performance, scalability, robustness and usability. Many attempts at tackling this issue have already been made [54, 60, 61, 57, 62, 23, 63, 64, 44], but a comprehensive comparison of TI methods across a large number of different datasets is still lacking. This is problematic, as new users to the field are confronted with an overwhelming choice of TI methods, without a clear idea of which would optimally solve their problem. Moreover, the strengths and weaknesses of existing methods need to be assessed, so that new developments in the field can focus on improving the current state-of-the-art.

In this study, we evaluated the accuracy, scalability, stability and usability of 45 TI methods (Figure 3.1a). We found substantial complementarity between current methods, with different sets of methods performing most optimally depending on the characteristics of the data. For method users, we created an interactive set of guidelines (available at guidelines.dynverse.org), which gives context-specific recommendations for method usage. Our evaluation also highlights some challenges for current methods, and our evaluation strategy can be useful to spearhead the development of new tools that accurately infer trajectories on ever more complex use cases.

3.2 Results

3.2.1 Trajectory inference methods

To make the outputs from different methods directly comparable to each other, we developed a common probabilistic model for representing trajectories from all possible sources (Figure 3.1b). In this

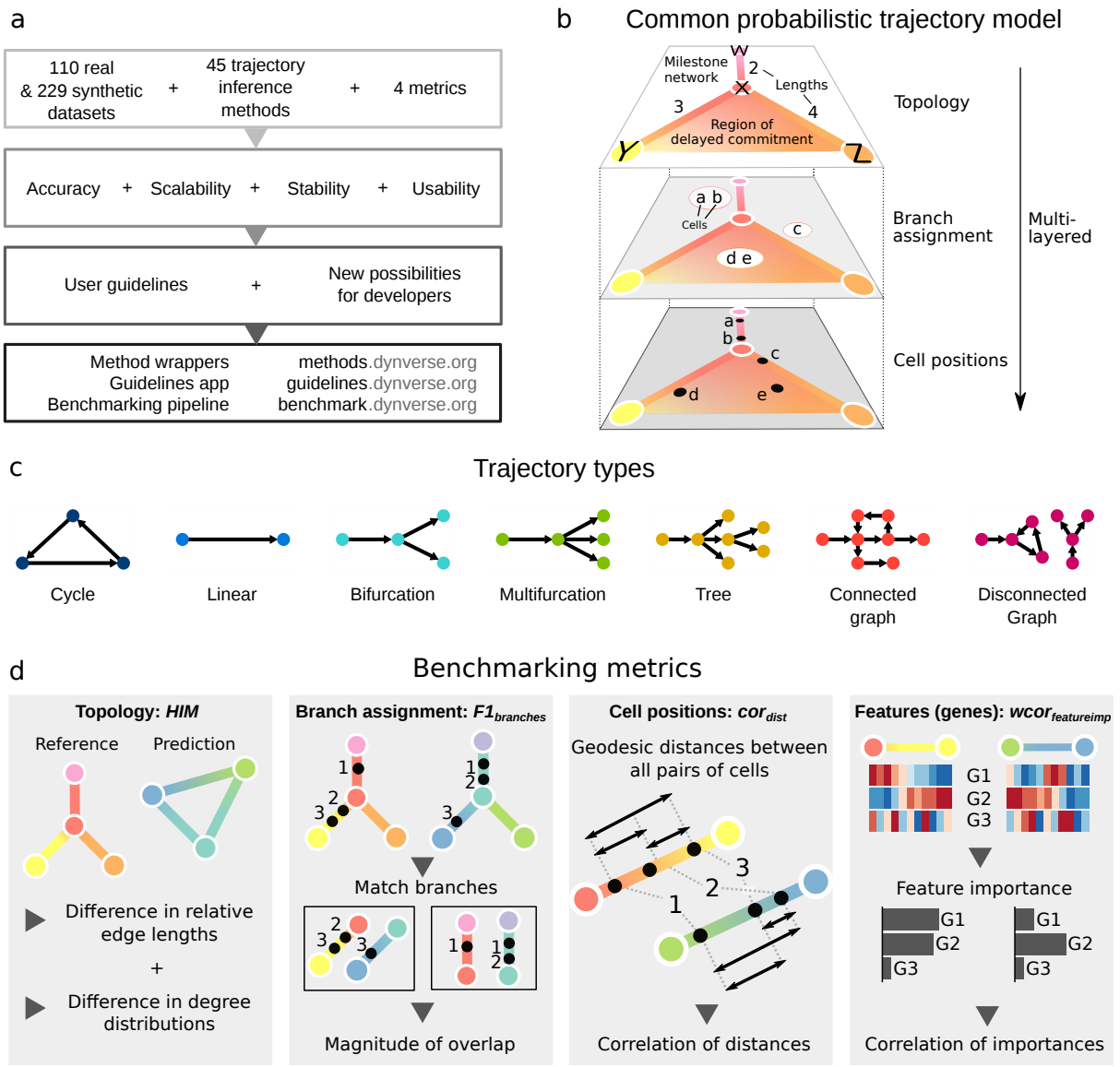


Figure 3.1: Overview of several key aspects of the evaluation. **a**, A schematic overview of our evaluation pipeline. **b**, To make the trajectories comparable to each other, a common trajectory model was used to represent reference trajectories from the real and synthetic datasets, as well as any predictions of TI methods. **c**, Trajectories are automatically classified into one of seven trajectory types, with increasing complexity. **d**, We defined four metrics, each assessing the quality of a different aspect of the trajectory. The HIM score assesses the similarity between the two topologies, taking into account differences in edge lengths and degree distributions. The F1branches assesses the similarity of the assignment of cells onto branches. The cordist quantifies the similarity in cellular positions between two trajectories, by calculating the correlation between pairwise geodesic distances. Finally, wcorfeatures quantifies the agreement between trajectory differentially expressed features from the known trajectory and the predicted trajectory.

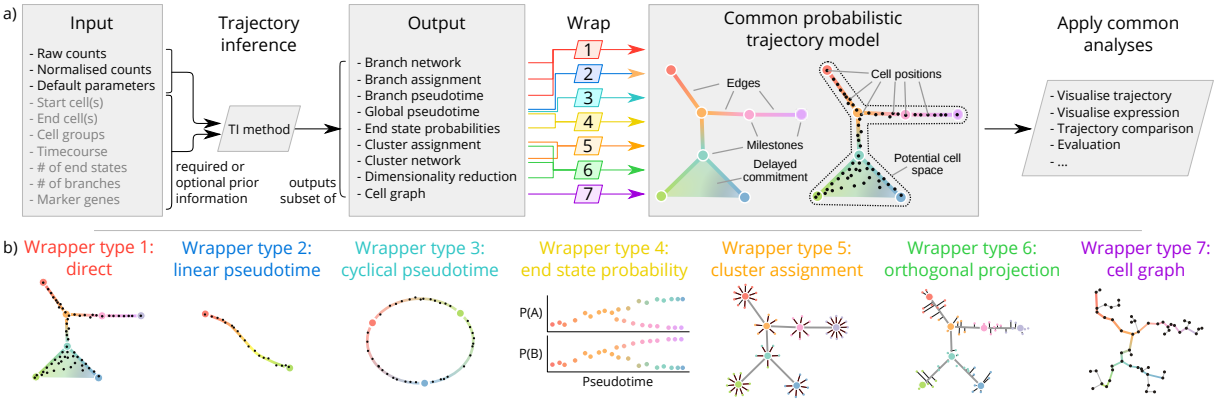


Figure 3.2: A common interface for TI methods. **a** The input and output of each TI method is standardized. As input, each TI method receives either raw or normalized counts, several parameters, and a selection of prior information. After its execution, a method uses one of the seven wrapper functions to transform its output to the common trajectory model. This common model then allows to perform common analysis functions on trajectory models produced by any TI method. **b** Illustrations of the specific transformations performed by each of the wrapper functions.

model, the overall topology is represented by a network of 'milestones', and the cells are placed within the space formed by each set of connected milestones. Although almost every method returned a unique set of outputs, we were able to classify these outputs into seven distinct groups (Figure 3.2) and we wrote a common output converter for each of these groups (Figure 3.3a). When strictly required, we also provided prior information to the method. These different priors can range from weak priors that are relatively easy to acquire, such as a start cell, to strong priors, such as a known grouping of cells, that are much harder to know a priori, and which can potentially introduce a large bias into the analysis (Figure 3.3a).

The largest difference between TI methods is whether a method fixes the topology and, if it does not, what kind of topology it can detect. We defined seven possible types of topology, ranging from very basic topologies (linear, cyclical and bifurcating) to the more complex ones (connected and disconnected graphs). Most methods either focus on inferring linear trajectories or limit the search to tree or less complex topologies, with only a selected few attempting to infer cyclic or disconnected topologies (Figure 3.3a).

We evaluated each method on four core aspects: (1) accuracy of a prediction, given a gold or silver standard on 110 real and 229 synthetic datasets; (2) scalability with respect to the number of cells and features (for example, genes); (3) stability of the predictions after subsampling the datasets; and (4) the usability of the tool in terms of software, documentation and the manuscript. Overall, we found a large diversity across the four evaluation criteria, with only a few methods, such as PAGA, Slingshot and SCORPIUS, performing well across the board (Figure 3.3b). We will discuss each evaluation criterion in more detail (Figure 3.4 and Supplementary Fig. 2), after which we conclude with guidelines for method users and future perspectives for method developers.

3.2.2 Accuracy

We defined several metrics to compare a prediction to a reference trajectory (Supplementary Note 1). Based on an analysis of their robustness and conformity to a set of rules (Supplementary Note 1), we chose four metrics each assessing a different aspect of a trajectory (Figure 3.1d): the topology (Hamming–Ipsen–Mikhailov, HIM), the quality of the assignment of cells to branches (F1branches), the cell positions (cordist) and the accuracy of the differentially expressed features along the trajectory (wcorfeatures). The data compendium consisted of both synthetic datasets, which offer the most

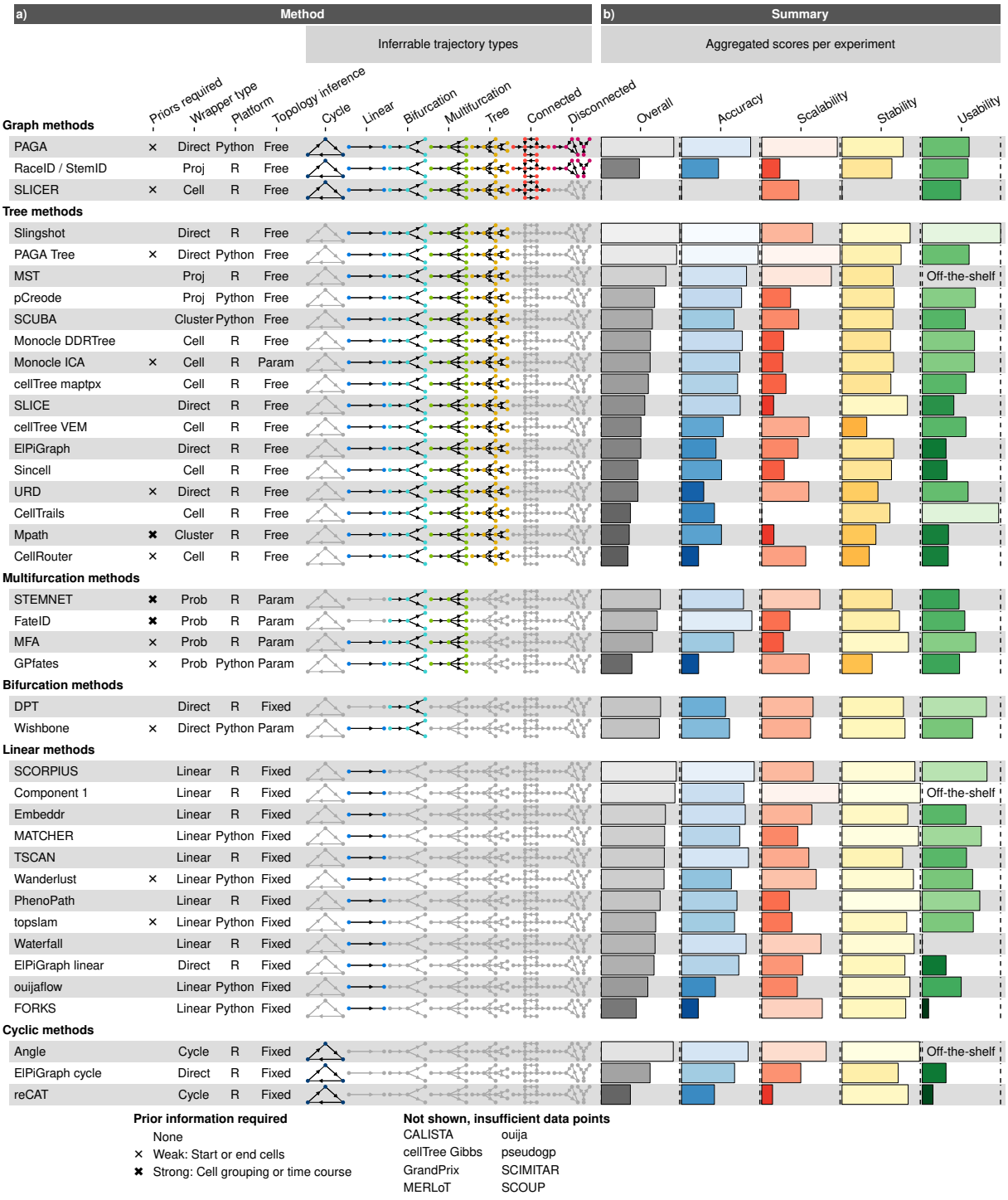


Figure 3.3: A characterization of the 45 methods evaluated in this study and their overall evaluation results. a, We characterized the methods according to the wrapper type, their required priors, whether the inferred topology is constrained by the algorithm (fixed) or a parameter (param), and the types of inferable topologies. The methods are grouped vertically based on the most complex trajectory type they can infer. **b**, The overall results of the evaluation on four criteria: accuracy using a reference trajectory on real and synthetic data, scalability with increasing number of cells and features, stability across dataset subsamples and quality of the implementation. Methods that errored on more than 50% of the datasets are not included in this figure and are shown instead in Supplementary Fig. 2.

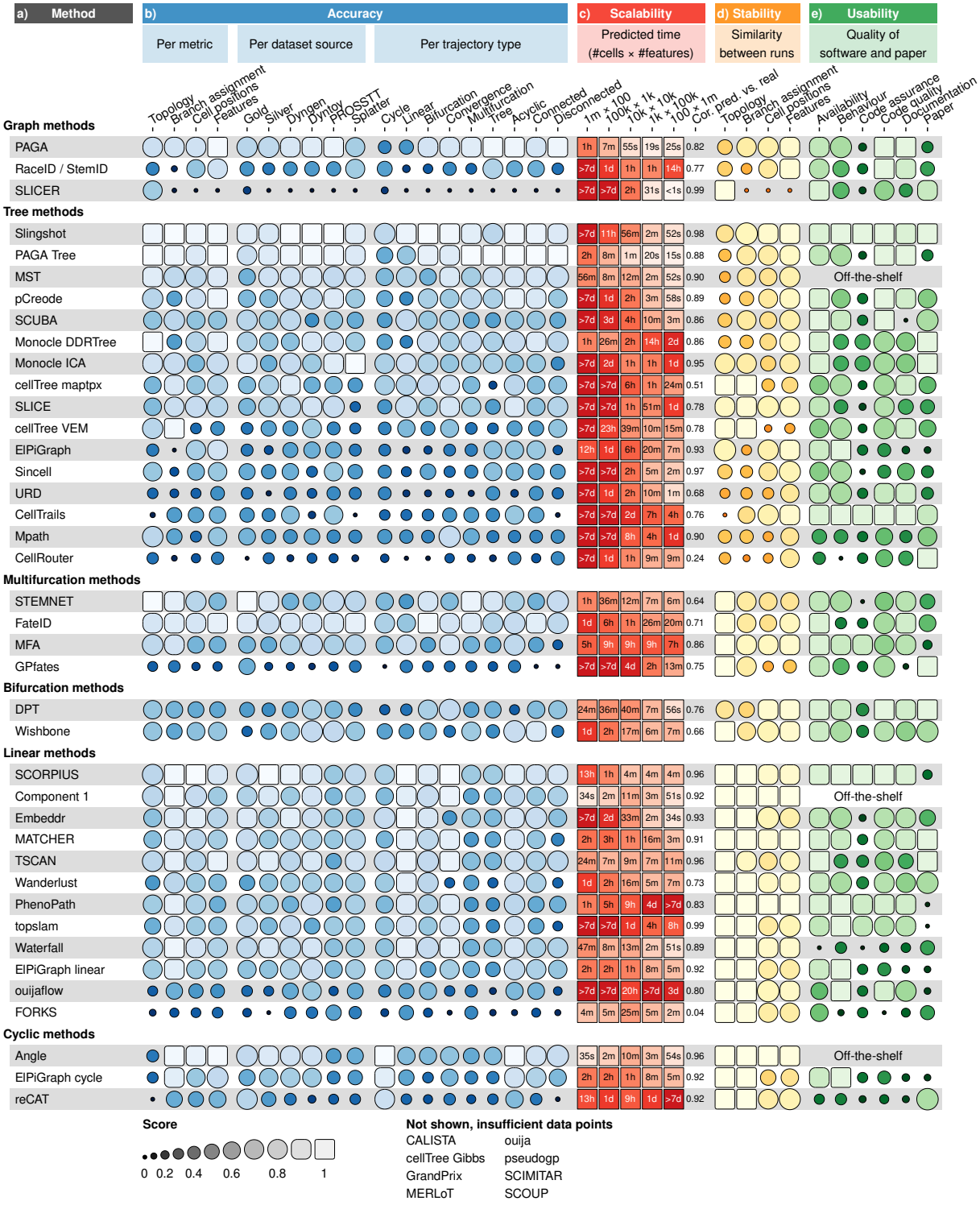


Figure 3.4: Detailed results of the four main evaluation criteria: accuracy, scalability, stability and usability. a, The names of the methods, ordered as in Figure 3.3. **b**, Accuracy of trajectory inference methods across metrics, dataset sources and dataset trajectory types. The performance of a method is generally more stable across dataset sources, but very variable depending on the metric and trajectory type. **c**, Predicted execution times for varying numbers of cells and features (no. of cells × no. of features). Predictions were made by training a regression model after running each method on bootstrapped datasets with varying numbers of cells and features. k, thousands; m, millions; cor, correlation. **d**, Stability results by calculating the average pairwise similarity between models inferred across multiple runs of the same method. **e**, Usability scores of the tool and corresponding manuscript, grouped per category. Off-the-shelf methods were directly implemented in R and thus do not have a usability score.

exact reference trajectory, and real datasets, which provide the highest biological relevance. These real datasets come from a variety of single-cell technologies, organisms and dynamic processes, and contain several types of trajectory topologies (Supplementary Table 2). Real datasets were classified as 'gold standard' if the reference trajectory was not extracted from the expression data itself, such as via cellular sorting or cell mixing [65]. All other real datasets were classified as 'silver standard'. For synthetic datasets we used several data simulators, including a simulator of gene regulatory networks using a thermodynamic model of gene regulation [66]. For each simulation, we used a real dataset as a reference, to match its dimensions, number of differentially expressed genes, drop-out rates and other statistical properties [38].

We found that method performance was very variable across datasets, indicating that there is no 'one-size-fits-all' method that works well on every dataset (Figure 3.5a). Even methods that can detect most of the trajectory types, such as PAGA, RaceID/StemID and SLICER were not the best methods across all trajectory types (Figure 3.4b). The overall score between the different dataset sources was moderately to highly correlated (Spearman rank correlation between 0.5–0.9) with the scores on real datasets containing a gold standard (Figure 3.5b), confirming both the accuracy of the gold standard trajectories and the relevance of the synthetic data. On the other hand, the different metrics frequently disagreed with each other, with Monocle and PAGA Tree scoring better on the topology scores, whereas other methods, such as Slingshot, were better at ordering the cells and placing them into the correct branches (Figure 3.4b).

The performance of a method was strongly dependent on the type of trajectory present in the data (Figure 3.4b). Slingshot typically performed better on datasets containing more simple topologies, while PAGA, pCreode and RaceID/StemID had higher scores on datasets with trees or more complex trajectories (Figure 3.5c). This was reflected in the types of topologies detected by every method, as those predicted by Slingshot tended to contain less branches, whereas those detected by PAGA, pCreode and Monocle DDRTree gravitated towards more complex topologies (Figure 3.5d). This analysis therefore indicates that detecting the right topology is still a difficult task for most of these methods, because methods tend to be either too optimistic or too pessimistic regarding the complexity of the topology in the data.

The high variability between datasets, together with the diversity in detected topologies between methods, could indicate some complementarity between the different methods. To test this, we calculated the likelihood of obtaining a top model when using only a subset of all methods. A top model in this case was defined as a model with an overall score of at least 95% as the best model. On all datasets, using one method resulted in getting a top model about 27% of the time. This increased up to 74% with the addition of six other methods (Figure 3.6a). The result was a relatively diverse set of methods, containing both strictly linear or cyclic methods, and methods with a broad trajectory type range such as PAGA. We found similar indications of complementarity between the top methods on data containing only linear, bifurcation or multifurcating trajectories (Figure 3.6b), although in these cases less methods were necessary to obtain at least one top model for a given dataset. Altogether, this shows that there is considerable complementarity between the different methods and that users should try out a diverse set of methods on their data, especially when the topology is unclear a priori. Moreover, it also opens up the possibilities for new ensemble methods that utilize this complementarity.

3.2.3 Scalability

While early TI methods were developed at a time where profiling more than a thousand cells was exceptional, methods now have to cope with hundreds of thousands of cells, and perhaps soon with

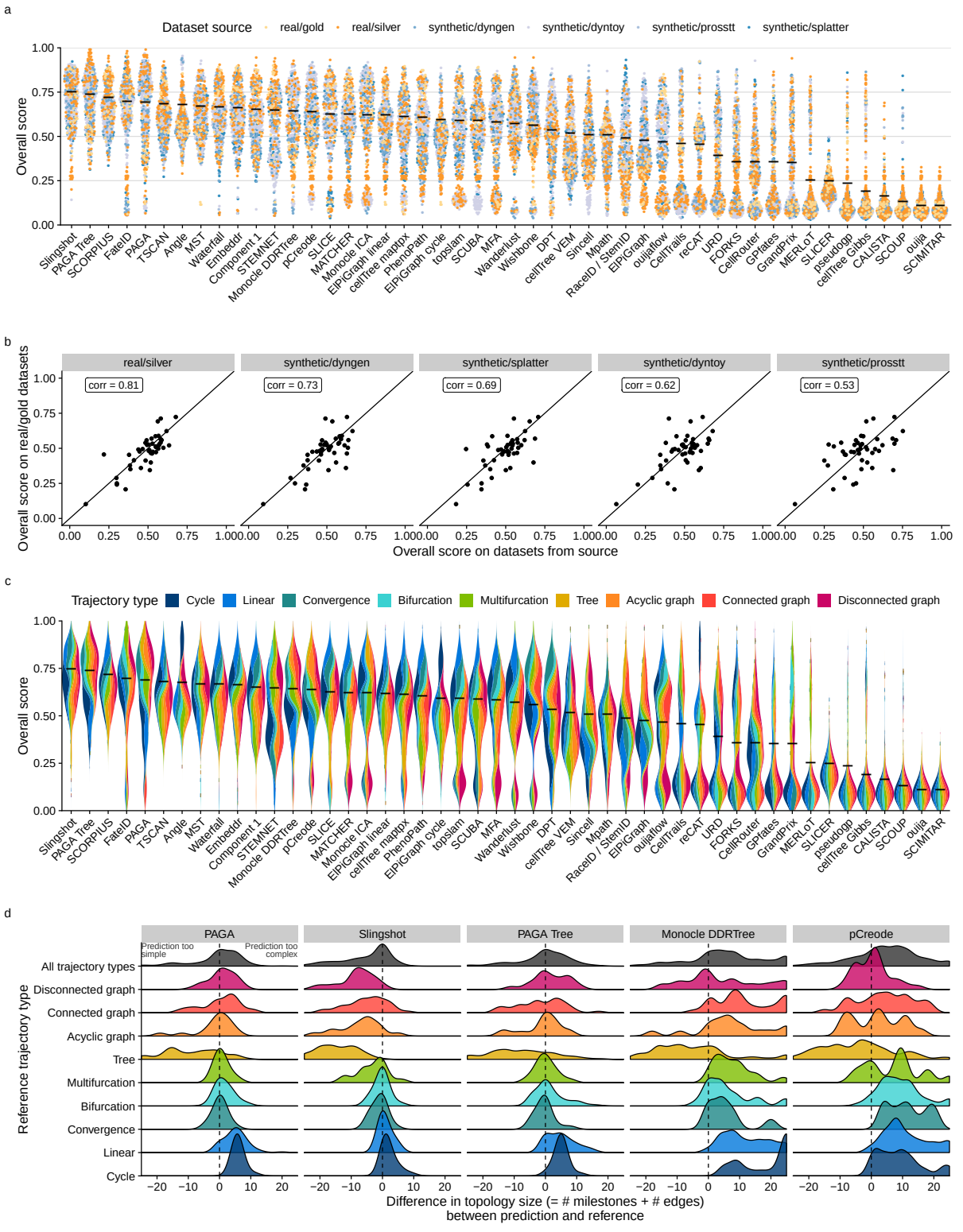


Figure 3.5: Accuracy of trajectory inference methods. **a** Overall score for all methods across 339 datasets, colored by the source of the datasets. Black line indicates the mean. **b** Similarity between the overall scores of all dataset sources, compared to real datasets with a gold standard, across all methods (n = 46, after filtering out methods that errored too frequently). Shown in the top left is the Pearson correlation. **c** Bias in the overall score towards trajectory types for all methods across 339 datasets. Black line indicates the mean. **d** Distributions of the difference in size between predicted and reference topologies. A positive difference means that the topology predicted by the method is more complex than the one in the reference.

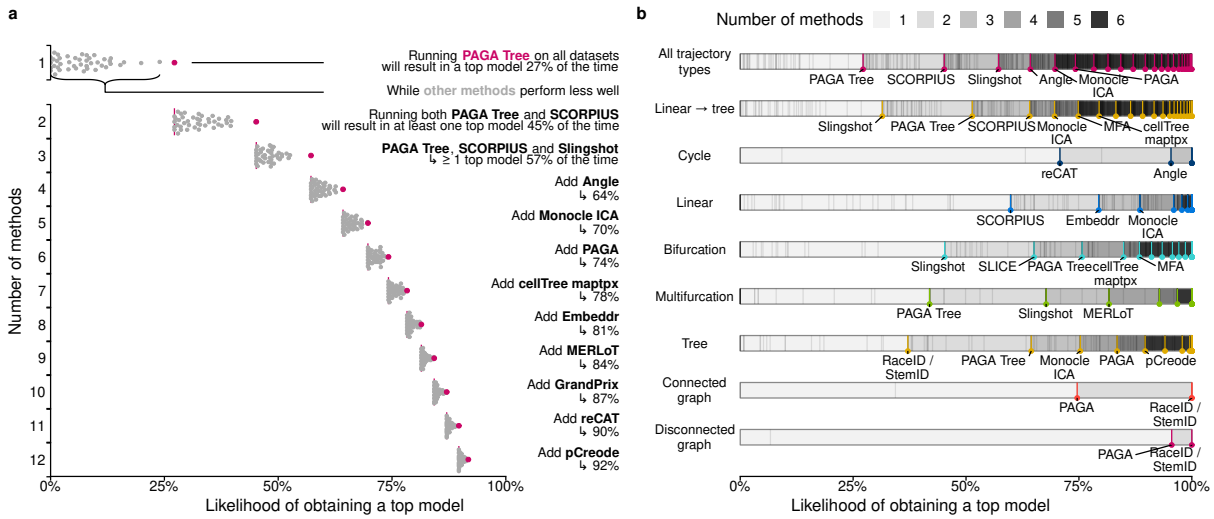


Figure 3.6: Complementarity between different trajectory inference methods. **a**, We assessed the likelihood for different combinations of methods to lead to a 'top model' (defined as a model with an overall score of at least 95% of the best model) when applied to all datasets. **b**, The likelihood for different combinations of methods to lead to a 'top model' was assessed separately on different trajectory types. For this figure, we did not include any methods requiring a cell grouping or a time course as prior information.

more than ten million [67]. Moreover, the recent application of TI methods on multi-omics single-cell data also showcases the increasing demands on the number of features [68]. To assess the scalability, we ran each method on up- and downscaled versions of five distinct real datasets. We modeled the running time and memory usage using a Shape Constrained Additive Model [69] (Figure 3.7a). As a control, we compared the predicted time (and memory) with the actual time (respectively memory) on all benchmarking datasets, and found that these were highly correlated overall (Spearman rank correlation >0.9, Supplementary Fig. 5), and moderately to highly correlated (Spearman rank correlation of 0.5–0.9) for almost every method, depending to what extent the execution of a method succeeded during the scalability experiments (Figure 3.4c and Supplementary Fig. 2a).

We found that the scalability of most methods was overall very poor, with most graph and tree methods not finishing within an hour on a dataset with ten thousand cells and ten thousand features (Figure 3.4c), which is around the size of a typical droplet-based single-cell dataset [67]. Running times increased further with increasing number of cells, with only a handful of graph/tree methods completing within a day on a million cells (PAGA, PAGA Tree, Monocle DDRTree, Stemnet and GrandPrix). Some methods, such as Monocle DDRTree and GrandPrix, also suffered from unsatisfactory running times when given a high number of features.

Methods with a low running time typically had two defining aspects: they had a linear time complexity with respect to the features and/or cells, and adding new cells or features led to a relatively low increase in time (Figure 3.7b). We found that more than half of all methods had a quadratic or superquadratic complexity with respect to the number of cells, which would make it difficult to apply any of these methods in a reasonable time frame on datasets with more than a thousand cells (Figure 3.7b).

We also assessed the memory requirements of each method (Supplementary Fig. 2c). Most methods had reasonable memory requirements for modern workstations or computer clusters ($\leq$12 GB) with PAGA and STEMNET in particular having a low memory usage with both a high number of cells or a high number of features. Notably, the memory requirements were very high for several methods on datasets with high numbers of cells (RaceID/StemID, pCreode and MATCHER) or features (Monocle DDRTree, SLICE and MFA).

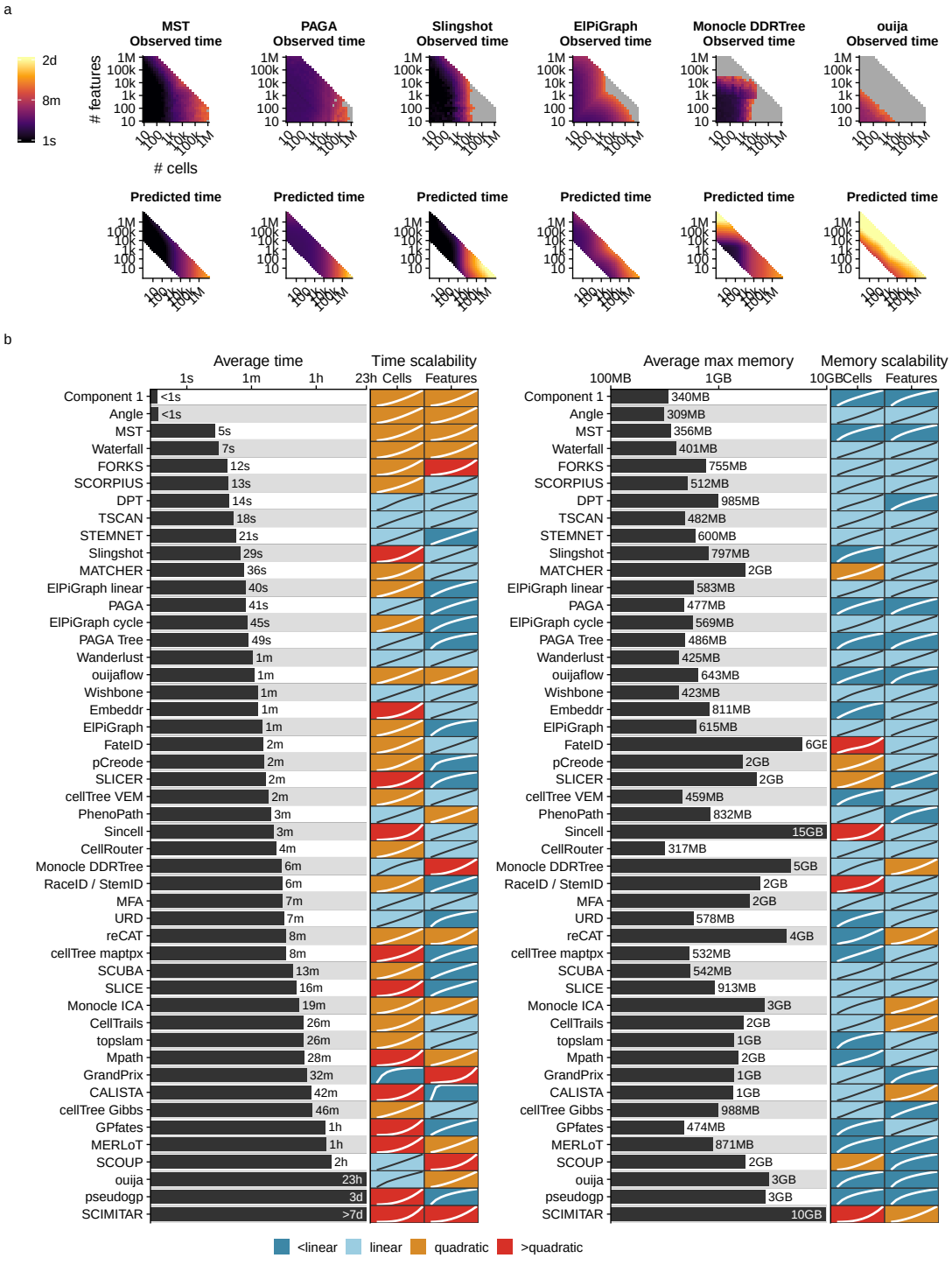


Figure 3.7: Scalability of trajectory inference methods. **a** Three examples of average observed running times across five datasets (left) and the predicted running time (right). **b** Overview of the scalability results of all methods, ordered by their average predicted running time from (a). We predicted execution times and memory usage for each method with increasing number of features or cells, and used these values to classify each method into sublinear, linear, quadratic and superquadratic based on the shape of the curve.

Altogether, the scalability analysis indicated that the dimensions of the data are an important factor in the choice of method, and that method development should pay more attention to maintaining reasonable running times and memory usage. §

3.2.4 Stability

It is not only important that a method is able to infer an accurate model in a reasonable time frame, but also that it produces a similar model when given very similar input data. To test the stability of each method, we executed each method on ten different subsamples of the datasets (95% of the cells, 95% of the features), and calculated the average similarity between each pair of models using the same scores used to assess the accuracy of a trajectory (Figure 3.4d).

Given that the trajectories of methods that fix the topology either algorithmically or through a parameter are already very constrained, it is to be expected that such methods tend to generate very stable results. Nonetheless, some fixed topology methods still produced slightly more stable results, such as SCORPIUS and MATCHER for linear methods and MFA for multifurcating methods. Stability was much more diverse among methods with a free topology. Slingshot produced more stable models than PAGA (Tree), which in turn produced more stable results than pCreode and Monocle DDRTree.

3.2.5 Usability

While not directly related to the accuracy of the inferred trajectory, it is also important to assess the quality of the implementation and how user-friendly it is for a biological user [70]. We scored each method using a transparent checklist of important scientific and software development practices, including software packaging, documentation, automated code testing and publication into a peer-reviewed journal (Table 3.1). It is important to note that there is a selection bias in the tools chosen for this analysis, as we did not include a substantial set of tools due to issues with installation, code availability and executability on a freely available platform (which excludes MATLAB). The reasons for not including certain tools are all discussed on our repository (https://github.com/dynverse/dynmethods/issues?q=label:un Installation issues seem to be quite general in bioinformatics [71] and the trajectory inference field is no exception.

We found that most methods fulfilled the basic criteria, such as the availability of a tutorial and elemental code quality criteria (Figure 3.4d and Supplementary Fig. 6). While recent methods had a slightly better quality score than older methods, several quality aspects were consistently lacking for the majority of the methods (Supplementary Fig. 6 right) and we believe that these should receive extra attention from developers. Although these outstanding issues covered all five categories, code assurance and documentation in particular were problematic areas, notwithstanding several studies pinpointing these as good practices [72, 73]. Only two methods had a nearly perfect usability score (Slingshot and Celltrails), and these could be used as an inspiration for future methods. We observed no clear relation between usability and method accuracy or usability (Figure 3.3b).

3.3 Discussion

In this study, we presented a large-scale evaluation of the performance of 45 TI methods. By using a common trajectory representation and four metrics to compare the methods' outputs, we were able to assess the accuracy of the methods on more than 200 datasets. We also assessed several other important quality measures, such as the quality of the method's implementation, the scalability to hundreds of thousands of cells and the stability of the output on small variations of the datasets.

Table 3.1: Scoring sheet for assessing usability of trajectory inference methods. Each quality aspect was given a weight based on how many times it was mentioned in a set of articles discussing best practices for tool development.

Aspect	Items	References
Availability		
Open source	(1) Method's code is freely available (2) The code can be run on a freely available platform	[74, 72, 70, 75, 73, 76, 77]
Version control	The code is available on a public version controlled repository, such as Github	[74, 72, 70, 75, 73, 76]
Packaging	(1) The code is provided as a "package", exposing functionality through functions or shell commands (2) The code can be easily installed through a repository such as CRAN, Bioconductor, PyPI, CPAN, debian packages, ...	[74, 75, 77, 76]
Dependencies	(1) Dependencies are clearly stated in the tutorial or in the code (2) Dependencies are automatically installed	[70, 75, 73, 78]
License	(1) The code is licensed (2) License allows academic use	[74, 70, 75, 73, 76, 77]
Interface	(1) The tool can be run using a graphical user interface, either locally or on a web server (2) The tool can be run through the command line or through a programming language	[76]
Code quality		
Function and object naming	(1) Functions/commands have well chosen names (2) Arguments/parameters have well chosen names	[72, 75]
Code style	(1) Code has a consistent style (2) Code follows (basic) good practices in the programming language of choice, for example PEP8 or the tidyverse style guide	[72, 75, 73]
Code duplication	Duplicated code is minimal	[72, 75]
Self-contained functions	The method is exposed to the user as self-contained functions or commands	[79, 70, 76]
Plotting	Plotting functions are provided for the final and/or intermediate results	
Dummy proofing	Package contains dummy proofing, i.e. testing whether the parameters and data supplied by the user make sense and are useful	[74, 78]
Code assurance		
Unit testing	Method is tested using unit tests	[74, 72, 79, 75, 76]
Unit testing	Tests are run automatically using functionality from the programming language	[74, 72, 79, 75, 76]
Continuous integration	The method uses continuous integration, for example on Travis CI	[80, 75, 73, 76]
Code coverage	(1) The code coverage of the repository is assessed. (2) What is the percentage of code coverage	
Documentation		
Support	(1) There is a support ticket system, for example on Github (2) The authors respond to tickets and issues are resolved within a reasonable time frame	[72, 75, 73, 76, 77]
Development model	(1) The repository separates the development code from master code, for example using git master en developer branches (2) The repository has created releases, or several branches corresponding to major releases. (3) The repository has branches for the development of separate features.	[81]
Tutorial	(1) A tutorial or vignette is available (2) The tutorial has example results (3) The tutorial has real example data (4) The tutorial showcases the method on several datasets (1=0, 2=0.5, >2=1)	[75, 76, 77, 78, 82]
Function documentation	(1) The purpose and usage of functions/commands is documented (2) The parameters of functions/commands are documented (3) The output of functions/commands is documented	[72, 70, 75, 76, 78]
Inline documentation	Inline documentation is present in the code	[72, 70, 75, 76, 78]
Parameter transparency	All important parameters are exposed to the user	[70]
Behaviour		
Seed setting	The method does not artificially become deterministic, for example by setting some (0.5) or a lot (1) of seeds	[83]
Unexpected output	(1) No unexpected output messages are generated by the method (2) No unexpected files, folders or plots are generated (3) No unexpected warnings during runtime or compilation are generated	[73]
Trajectory format	The postprocessing necessary to extract the relevant output from the method is minimal (1), moderate (0.5) or extensive (0)	
Prior information	Prior information is required (0), optional (1) or not required (1)	
Paper		
Publishing	The method is published	
Peer review	The paper is published in a peer-reviewed journal	[78, 84, 85]
Evaluation on real data	(1) The paper shows the method's usefulness on several (1), one (0.25) or no real datasets. (2) The paper quantifies the accuracy of the method given a gold or silver standard trajectory	[86, 87]
Evaluation of robustness	The paper assessed method robustness (to eg. noise, subsampling, parameter changes, stability) in one (0.5) or several (1) ways	[78, 86, 82, 87]

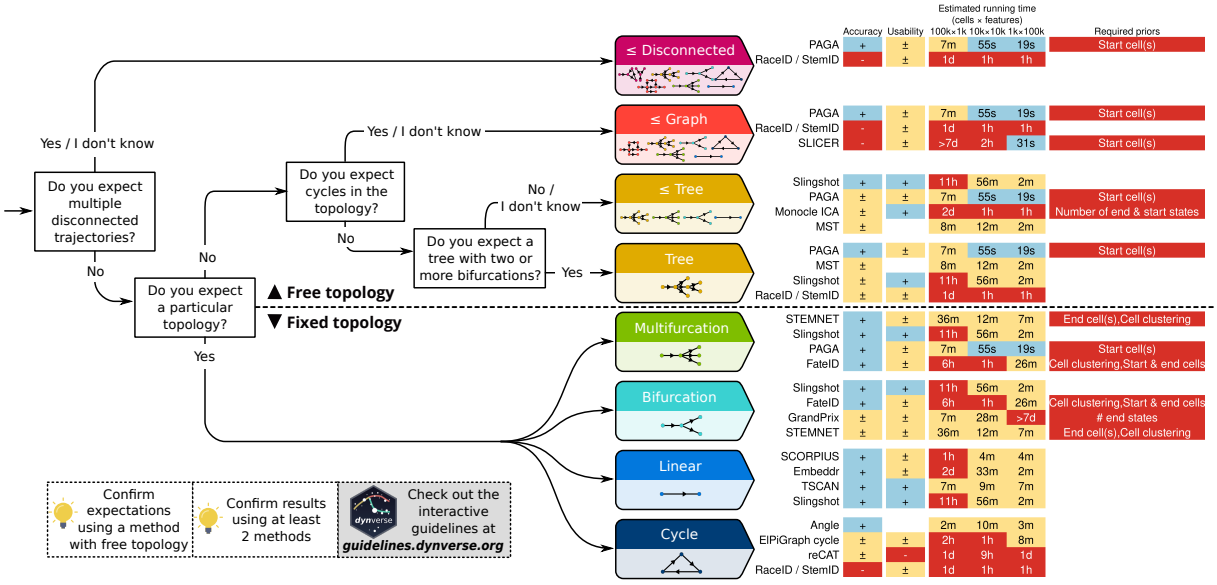


Figure 3.8: Practical guidelines for method users. As the performance of a method mostly depends on the topology of the trajectory, the choice of TI method will be primarily influenced by the user's existing knowledge about the expected topology in the data. We therefore devised a set of practical guidelines, which combines the method's performance, user friendliness and the number of assumptions a user is willing to make about the topology of the trajectory. Methods to the right are ranked according to their performance on a particular (set of) trajectory type. Further to the right are shown the accuracy (+: scaled performance $\geq$ 0.9, $\pm$: >0.6), usability scores (+:$\geq$0.9, $\pm$ $\geq$0.6), estimated running times and required prior information. k, thousands; m, millions.

Based on the results of our benchmark, we propose a set of practical guidelines for method users (Figure 3.8 and guidelines.dynverse.org). We postulate that, as a method's performance is heavily dependent on the trajectory type being studied, the choice of method should currently be primarily driven by the anticipated trajectory topology in the data. For most use cases, the user will know very little about the expected trajectory, except perhaps whether the data is expected to contain multiple disconnected trajectories, cycles or a complex tree structure. In each of these use cases, our evaluation suggests a different set of optimal methods, as shown in Figure 3.8. Several other factors will also impact the choice of methods, such as the dimensions of the dataset and the prior information that is available. These factors and several others can all be dynamically explored in our interactive app (guidelines.dynverse.org). This app can also be used to query the results of this evaluation, such as filtering the datasets or changing the importance of the evaluation metrics for the final ranking.

When inferring a trajectory on a dataset of interest, it is important to take two further points into account. First, it is critical that a trajectory, and the downstream results and/or hypotheses originating from it, are confirmed by multiple TI methods. This is to make sure that the prediction is not biased due to the given parameter setting or the particular algorithm underlying a TI method. The value of using different methods is further supported by our analysis indicating substantial complementarity between the different methods. Second, even if the expected topology is known, it can be beneficial to also try out methods that make less assumptions about the trajectory topology. When the expected topology is confirmed using such a method, it provides additional evidence to the user. When a more complex topology is produced, this could indicate that the underlying biology is much more complex than anticipated by the user.

Critical to the broad applicability of TI methods is the standardization of the input and output interfaces of TI methods, so that users can effortlessly execute TI methods on their dataset of interest,

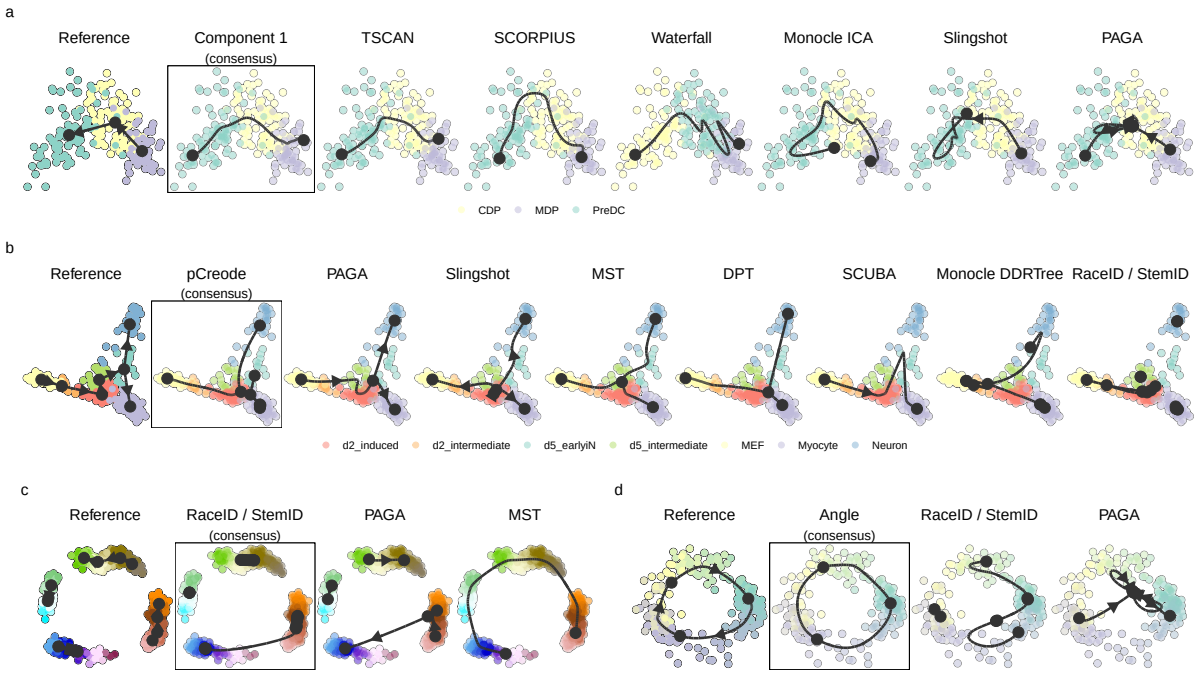


Figure 3.9: Demonstration of how a common framework for TI methods facilitates broad applicability using some example datasets. Trajectories inferred by each method were projected to a common dimensionality reduction using multidimensional scaling. For each dataset, we also calculated a 'consensus' prediction, by calculating the cordist between each pair of models and picking the model with the highest score on average. **a**, The top methods applied on a dataset containing a linear trajectory of differentiation dendritic cells, going from MDP, CDP to PreDC. **b**, The top methods applied on a dataset containing a bifurcating trajectory of reprogrammed fibroblasts. **c**, A synthetic dataset generated by dyntoy, containing four disconnected trajectories. **d**, A synthetic dataset generated by dyngen, containing a cyclic trajectory.

compare different predicted trajectories and apply downstream analyses, such as finding genes important for the trajectory, network inference [27] or finding modules of genes [88]. Our framework is an initial attempt at tackling this problem, and we illustrate its usefulness here by comparing the predicted trajectories of several top-performing methods on datasets containing a linear, tree, cyclic and disconnected graph topology (Figure 3.9). Using our framework, this figure can be recreated using only a couple of lines of R code (https://methods.dynverse.org). In the future, this framework could be extended to allow additional input data, such as spatial and RNA velocity information [89], and easier downstream analyses. In addition, further discussion within the field is required to arrive at a consensus concerning a common interface for trajectory models, which can include additional features such as uncertainty and gene importance.

Our study indicates that the field of trajectory inference is maturing, primarily for linear and bifurcating trajectories (Figure 3.9a,b). However, we also highlight several ongoing challenges, which should be addressed before TI can be a reliable tool for analyzing single-cell omics datasets with complex trajectories. Foremost, new methods should focus on improving the unbiased inference of tree, cyclic graph and disconnected topologies, as we found that methods repeatedly overestimate or underestimate the complexity of the underlying topology, even if the trajectory could easily be identified using a dimensionality reduction method (Figure 3.9c,d). Furthermore, higher standards for code assurance and documentation could help in adopting these tools across the single-cell omics field. Finally, new tools should be designed to scale well with the increasing number of cells and features. We found that only a handful of current methods can handle datasets with more than 10,000 cells within a reasonable time frame. To support the development of these new tools, we provide a

series of vignettes on how to wrap and evaluate a method on the different measures proposed in this study at https://benchmark.dynverse.org.

We found that the performance of a method can be very variable between datasets, and therefore included a large set of both real and synthetic data within our evaluation, leading to a robust overall ranking of the different methods. However, 'good-yet-not-the-best' methods [90] can still provide a very valuable contribution to the field, especially if they make use of novel algorithms, return a more scalable solution or provide a unique insight in specific use cases. This is also supported by our analysis of method complementarity. Some examples for the latter include PhenoPath, which can include additional covariates in its model, ouija, which returns a measure of uncertainty of each cell's position within the trajectory, and StemID, which can infer the directionality of edges within the trajectory.

3.4 Methods

3.4.1 Trajectory inference methods

We gathered a list of 71 trajectory inference tools (Supplementary Table 1) by searching the literature for 'trajectory inference' and 'pseudotemporal ordering', and based on two existing lists found online: https://github.com/seandavi/awesome-single-cell [49] and https://github.com/agitter/single-cell-pseudotime [91]. We welcome any contributions by creating an issue at https://methods.dynverse.org.

Methods were excluded from the evaluation based on several criteria: (1) not freely available; (2) no code available; (3) superseded by another method; (4) requires data types other than expression; (5) no programming interface; (6) unresolved errors during wrapping; (7) too slow (requires more than 1 h on a 100 $\times$ 100 dataset); (8) does not return an ordering; and (9) requires additional user input during the algorithm (other than prior information). The discussions on why these methods were excluded can be found at https://github.com/dynverse/dynmethods/issues?q=label:unwrappable. In the end, we included 45 methods in the evaluation.

3.4.2 Method wrappers

To make it easy to run each method in a reproducible manner, each method was wrapped within Docker and singularity containers (available at https://methods.dynverse.org). These containers are automatically built and tested using Travis continuous integration (https://travis-ci.org/dynverse) and can be ran using both Docker and Singularity. For each method, we wrote a wrapper script based on example scripts or tutorials provided by the authors (as mentioned in the respective wrapper scripts). This script reads in the input data, runs the method and outputs the files required to construct a trajectory. We also created a script to generate an example dataset, which is used for automated testing.

We used the Github issues system to contact the authors of each method, and asked for feedback on the wrappers, the metadata and the usability scores. About one-third of the authors responded and we improved the wrappers based on their feedback. These discussions can be viewed on Github: https://github.com/dynverse/dynmethods/issues?q=label:method_discussion

Method input

As input, we provided each method with either the raw count data (after cell and gene filtering) or normalized expression values, based on the description in the method documentation or from the study describing the method. A large portion of the methods requires some form of prior information

(for example, a start cell) to be executable. Other methods optionally allow the exploitation of certain prior information. Prior information can be supplied as a starting cell from which the trajectory will originate, a set of important marker genes or even a grouping of cells into cell states. Providing prior information to a TI method can be both a blessing and a curse. In one way, prior information can help the method to find the correct trajectory among many, equally likely, alternatives. On the other hand, incorrect or noisy prior information can bias the trajectory towards current knowledge. Moreover, prior information is not always easily available, and its subjectivity can therefore lead to multiple equally plausible solutions, restricting the applicability of such TI methods to well-studied systems.

The prior information was extracted from the reference trajectory as follows:

- **Start cells**: the identity of one or more start cells. For both real and synthetic data, a cell was chosen that was the closest (in geodesic distance) to each milestone with only outgoing edges. For ties, one random cell was chosen. For cyclic datasets, a random cell was chosen.
- **End cells**: the identity of one or more end cells. This is similar to the start cells, but now for every state with only incoming edges.
- **No. of end states**: number of terminal states, i.e., the number of milestones with only incoming edges.
- **Grouping**: for each cell a label showing which state/cluster/branch it belongs to. For real data, the states were from the gold/silver standard. For synthetic data, each milestone was seen as one group and cells were assigned to their closest milestone.
- **No. of branches**: number of branches/intermediate states. For real data, this was the number of states in the gold/silver standard. For synthetic data, this was the number of milestones.
- **Discrete time course**: for each cell a time point from which it was sampled. If available, this was directly extracted from the reference trajectory; otherwise the geodesic distance from the root milestone was used. For synthetic data, the simulation time was uniformly discretized into four timepoints.
- **Continuous time course**: for each cell a time point from which it was sampled. For real data, this was equal to the discrete time course. For synthetic data, we used the internal simulation time of each simulator.

Common trajectory model

Due to the absence of a common format for trajectory models, most methods return a unique set of output formats with few overlaps. We therefore post-processed the output of each method into a common probabilistic trajectory model (Figure 3.2a). This model consisted of three parts. (1) The milestone network represents the overall network topology, and contains edges between different milestones and the length of the edge between them. (2) The milestone percentages contain, for each cell, its position between milestones and sums for each cell to one. (3) The regions of delayed commitment define connections between three or more milestones. These must be explicitly defined in the trajectory model and per region one milestone must be directly connected to all other milestones of the region.

Depending on the output of a method, we used different strategies to convert the output to our model (Figure 3.2b). Special conversions are denoted by an asterisk and will be explained in more detail in the second list below.

- **Type 1, direct**: CALISTA*, DPT*, ElPiGraph, ElPiGraph cycle, ElPiGraph linear, MERLoT, PAGA, SLICE*, Slingshot, URD* and Wishbone. The wrapped method directly returned a network of milestones, the regions of delayed commitment and for each cell it is given to what extent it belongs to a milestone. In some cases, this indicates that additional transformations were required for the method, not covered by any of the following output formats. Some methods returned a branch network instead of a milestone network and this network was converted by calculating the line graph of the branch network.
- **Type 2, linear pseudotime**: Component 1, Embeddr, FORKS, MATCHER, ouija, ouijaflow, PhenoPath, pseudogp, SCIMITAR, SCORPIUS, topslam, TSCAN, Wanderlust and Waterfall. The method returned a pseudotime, which is translated into a linear trajectory where the milestone network contains two milestones and cells are positioned between these two milestones.
- **Type 3, cyclical pseudotime**: Angle and reCAT. The method returned a pseudotime, which is translated into a cyclical trajectory where the milestone network contains three milestones and cells are positioned between these three milestones. These milestones were positioned at pseudotime 0, 1/3 and 2/3.
- **Type 4, end state probability**: FateID, GPfates, GrandPrix, MFA*, SCOUP and STEMNET. The method returned a pseudotime and for each cell and end state a probability (Pr) for how likely a cell will end up in a certain end state. This was translated into a star-shaped milestone network, with one starting milestone (M0) and several outer milestones (Mi), with regions of delayed commitment between all milestones. The milestone percentage of a cell to one of the outer milestones was equal to pseudotime$\times$PrMi. The milestone percentage to the starting milestone was equal to $1 -$ pseudotime.
- **Type 5, cluster assignment**: Mpath and SCUBA. The method returned a milestone network and an assignment of each cell to a specific milestone. Cells were positioned onto the milestones they are assigned to, with milestone percentage equal to 1.
- **Type 6, orthogonal projection**: MST, pCreode and RaceID/StemID. The method returned a milestone network, and a dimensionality reduction of the cells and milestones. The cells were projected to the closest nearest segment, thus determining the cells' position along the milestone network. If a method also returned a cluster assignment (type 5), we limited the projection of each cell to the closest edge connecting to the milestone of a cell. For these methods, we usually wrote two wrappers, one which included the projection and one without.
- **Type 7, cell graph**: CellRouter, CellTrails, cellTree Gibbs, cellTree maptpx, cellTree VEM, Monocle DDRTree, Monocle ICA, Sincell* and SLICER. The method returned a network of cells and which cell–cell transitions were part of the 'backbone' structure. Backbone cells with degree $\neq$ 2 were regarded as milestones and all other cells were placed on transitions between the milestones. If a method did not return a distance between pairs of cells, the cells were uniformly positioned between the two milestones. Otherwise, we first calculated the distance between two milestones as the sum of the distances between the cells and then divided the distance of each pair of cells with the total distance to get the milestone percentages.

Special conversions were necessary for certain methods:

- **CALISTA**: We assigned the cells to the branch at which the sum of the cluster probabilities of two connected milestones was the highest. The cluster probabilities of the two selected milestones were then used as milestone percentages. This was then processed as a type 1, direct, method.

- **DPT**: We projected the cells onto the cluster network, consisting of a central milestone (this cluster contained the cells that were assigned to the 'unknown' branch) and three terminal milestones, each corresponding to a tip point. This was then processed as a type 1, direct, method.
- **Sincell**: To constrain the number of milestones this method creates, we merged two cell clusters iteratively until the percentage of leaf nodes was below a certain cutoff, with the default cutoff set to 25%. This was then processed as a type 7, cell graph, method.
- **SLICE**: As discussed in the vignette of SLICE (https://research.cchmc.org/pbge/slice.html), we ran principal curves one by one for every edge detected by SLICE. This was then processed as a type 1, direct, method.
- **MFA**: We used the branch assignment as state probabilities, which together with the global pseudotime were processed as a type 4, end state probabilities, method.
- **URD**: We extracted the pseudotime of a cell within each branch using the y positions in the tree layout. This was then further processed as a type 1, direct, method.

More information on how each method was wrapped can be found within the comments of each wrapper script, listed at https://methods.dynverse.org.

Off-the-shelf methods

For baseline performance, we added several 'off-the-shelf' TI methods that can be run using a few lines of code in R.

- **Component 1**: This method returns the first component of a principal component analysis (PCA) dimensionality reduction as a linear trajectory. This method is especially relevant as it has been used in a few studies already [92, 93].
- **Angle**: Similar to the previous method, this method computes the angle with respect to the origin in a two-dimensional PCA and uses this angle as a pseudotime for generating a cyclical trajectory.
- **MST**: This method performs PCA dimensionality reduction, followed by clustering using the R mclust package, after which the clusters are connected using a minimum spanning tree. The trees are orthogonally projected to the nearest segment of the tree. This baseline is highly relevant as many methods follow the same methodology: dimensionality reduction, clustering, topology inference and project cells to topology.

3.4.3 Trajectory types

We classified all possible trajectory topologies into distinct trajectory types, based on topological criteria (Figure 3.1c). These trajectory types start from the most general trajectory type, a disconnected graph, and move down (within a directed acyclic graph structure), progressively becoming more simple until the two basic types are reached: linear and cyclical. A disconnected graph is a graph in which only one edge can exist between two nodes. A (connected) graph is a disconnected graph in which all nodes are connected. An acyclic graph is a graph containing no cycles. A tree is an acyclic graph containing no convergences (no nodes with in-degree higher than 1). A convergence is an acyclic graph in which only one node has a degree larger than 1 and this same node has an in-degree of 1. A multifurcation is a tree in which only one node has a degree larger than 1. A bifurcation is a multifurcation in which only one node has a degree equal to 3. A linear topology is a graph in which

no node has a degree larger than 3. Finally, a cycle is a connected graph in which every node has a degree equal to 2. In most cases, a method that was able to detect a complex trajectory type was also able to detect less complex trajectory types, with some exceptions shown in Figure 3.3a.

For simplicity, we merged the bifurcation and convergence trajectory type, and the acyclic graph and connected graph trajectory type in the main figures of the paper.

3.4.4 Real datasets

We gathered real datasets by searching for 'single-cell' at the Gene Expression Omnibus and selecting those datasets in which the cells are sampled from different stages in a dynamic process (Supplementary Table 2). The scripts to download and process these datasets are available on our repository (https://benchmark.dynverse.org/tree/master/scripts/01-datasets). Whenever possible, we preferred to start from the raw counts data. These raw counts were all normalized and filtered using a common pipeline, as discussed later. Some original datasets contained more than one trajectory, in which case we split the dataset into its separate connected trajectory, but also generated several combinations of connected trajectories to include some datasets with disconnected trajectories in the evaluation. In the end, we included 110 datasets for this evaluation.

For each dataset, we extracted a reference trajectory, consisting of two parts: the cellular grouping (milestones) and the connections between these groups (milestone network). The cellular grouping was provided by the authors of the original study, and we classified it as a gold standard when it was created independently from the expression matrix (such as from cell sorting, the origin of the sample, the time it was sampled or cellular mixing) or as a silver standard otherwise (usually by clustering the expression values). To connect these cell groups, we used the original study to determine the network that the authors validated or otherwise found to be the most likely. In the end, each group of cells was placed on a milestone, having a percentage of 1 for that particular milestone. The known connections between these groups were used to construct the milestone network. If there was biological or experimental time data available, we used this as the length of the edge; otherwise we set all the lengths equal to one.

3.4.5 Synthetic datasets

To generate synthetic datasets, we used four different synthetic data simulators:

- **dyngen**: simulations of gene regulatory networks, available at https://github.com/dynverse/dyngen
- **dyntoy**: random gradients of expression in the reduced space, available at https://github.com/dynverse/dyntoy
- **PROSSTT**: expression is sampled from a linear model that depends on pseudotime [39]
- **Splatter**: simulations of non-linear paths between different expression states [38]

For every simulator, we took great care to make the datasets as realistic as possible. To do this, we extracted several parameters from all real datasets. We calculated the number of differentially expressed features within a trajectory using a two-way Mann–Whitney U test between every pair of cell groups. These values were corrected for multiple testing using the Benjamini-Hochberg procedure (FDR < 0.05) and we required that a gene was expressed in at least 5% of cells, and had at least a fold-change of 2. We also calculated several other parameters, such as drop-out rates and library sizes using the Splatter package [38]. These parameters were then given to the simulators when applicable, as described for each simulator below. Not every real dataset was selected to serve as a reference for a synthetic dataset. Instead, we chose a set of ten distinct reference real datasets by clustering all

the parameters of each real dataset, and used the reference real datasets at the cluster centers from a pam clustering (with $k = 10$, implemented in the R cluster package) to generate synthetic data.

dyngen

The dyngen (https://github.com/dynverse/dyngen) workflow to generate synthetic data is based on the well established workflow used in the evaluation of network inference methods [66, 30] and consists of four main steps: network generation, simulation, gold standard extraction and simulation of the scRNA-seq experiment. At every step, we tried to mirror real regulatory networks, while keeping the model simple and easily extendable. We simulated a total of 110 datasets, with 11 different topologies.

Network generation

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 genes [94] strongly repress each other, forcing one of the two to become inactive [95]. Such mutual antagonism can be modelled and simulated [96, 97]. Although such a two-gene model is simple and elegant, the reality is frequently more complex, with multiple genes (grouped into modules) repressing each other [98].

To simulate certain trajectory topologies, we therefore designed module networks in which the cells follow a particular trajectory topology given certain parameters. Two module networks generated linear trajectories (linear and linear long), one generated a bifurcation, one generated a convergence, one generated a multifurcation (trifurcating), two generated a tree (consecutive bifurcating and binary tree), one generated an acyclic graph (bifurcating and converging), one generated a complex fork (trifurcating), one generated a rooted tree (consecutive bifurcating) and two generated simple graph structures (bifurcating loop and bifurcating cycle). The structure of these module networks is available at https://github.com/dynverse/dyngen/tree/master/inst/ext_data/modulenetworks.

From these module networks we generated gene regulatory networks in two steps: the main regulatory network was first generated, and extra target genes from real regulatory networks were added. For each dataset, we used the same number of genes as were differentially expressed in the real datasets. 5% of the genes were assigned to be part of the main regulatory network, and were randomly distributed among all modules (with at least one gene per module). We sampled edges between these individual genes (according to the module network) using a uniform distribution between 1 and the number of possible targets in each module. To add additional target genes to the network, we assigned every regulator from the network to a real regulator in a real network (from regulatory circuits [99]), and extracted for every regulator a local network around it using personalized pagerank (with damping factor set to 0.1), as implemented in the page_rank function of the *igraph* package.

Simulation of gene regulatory systems using thermodynamic models

To simulate the gene regulatory network, we used a system of differential equations similar to those used in evaluations of gene regulatory network inference methods [30]. In this model, the changes in gene expression (x_i) and protein expression (y_i) are modeled using ordinary differential

equations [66] (ODEs):

$$\frac{dx_i}{dt} = \underbrace{m \times f(y_1, y_2, ...)}_{\text{production}} - \underbrace{\lambda \times x_i}_{\text{degradation}}$$

$$\frac{dy_i}{dt} = \underbrace{r \times x_i}_{\text{production}} - \underbrace{\Lambda \times y_i}_{\text{degradation}}$$

where m, λ, r and Λ represent production and degradation rates, the ratio of which determines the maximal gene and protein expression. The two types of equations are coupled because the production of protein y_i depends on the amount of gene expression x_i, which in turn depends on the amount of other proteins through the activation function $f(y_1, y_2, ...)$.

The activation function is inspired by a thermodynamic model of gene regulation, in which the promoter of a gene can be bound or unbound by a set of transcription factors, each representing a certain state of the promoter. Each state is linked with a relative activation α_j, a number between 0 and 1 representing the activity of the promoter at this particular state. The production rate of the gene is calculated by combining the probabilities of the promoter being in each state with the relative activation:

$$f(y_1, y_2, ..., y_n) = \sum_{j \in \{0,1,...,n^2\}} \alpha_j \times P_j$$

The probability of being in a state is based on the thermodynamics of transcription factor binding. When only one transcription factor is bound in a state:

$$P_j \propto \nu = \left(\frac{y}{k}\right)^n$$

where the hill coefficient n represents the cooperativity of binding and k the transcription factor concentration at half-maximal binding. When multiple regulators are bound:

$$P_j \propto \nu = \rho \times \prod_j \left(\frac{y_j}{k_j}\right)^{n_j}$$

where ρ represents the cooperativity of binding between the different transcription factors.

P_i is only proportional to ν because ν is normalized such that $\sum_i P_i = 1$.

To each differential equation, we added an additional stochastic term:

$$\frac{dx_i}{dt} = m \times f(y_1, y_2, ...) - \lambda \times x_i + \eta \times \sqrt{x_i} \times \Delta W_t$$
$$\frac{dy_i}{dt} = r \times x_i - \Lambda \times y_i + \eta \times \sqrt{y_i} \times \Delta W_t$$

with $\Delta W_t \sim \mathcal{N}(0, h)$.

Similar to GeneNetWeaver [66], we sample the different parameters from random distributions, defined as follows. e defines whether a transcription factor activates (1) or represses (-1), as defined within the regulatory network network.

$$
\begin{aligned}
r &= \mathcal{U}(10, 200) \\
d &= \mathcal{U}(2, 8) \\
p &= \mathcal{U}(2, 8) \\
q &= \mathcal{U}(1, 5) \\
a_0 &= \begin{cases} 1 & \text{if } |e| = 0 \\ 1 & \text{if } \forall x \in e,\ x = -1 \\ 0 & \text{if } \forall x \in e,\ x = 1 \\ 0.5 & \text{otherwise} \end{cases} \\
a_i &= \begin{cases} 0 & \text{if } \exists x \in e_i,\ x = -1 \\ 1 & \text{otherwise} \end{cases} \\
s &= \mathcal{U}(1, 20) \\
k &= y_{max}/(2 * s), \\
&\quad \text{where } y_{max} = r/d \times p/q \\
c &= \mathcal{U}(1, 4)
\end{aligned}
$$

We converted each ODE to an SDE by adding a chemical Langevin equation, as described in [66]. These SDEs were simulated using the Euler–Maruyama approximation, with time-step $h = 0.01$ and noise strength $\eta = 8$. The total simulation time varied between 5 for linear and bifurcating datasets, 10 for consecutive bifurcating, trifurcating and converging datasets, 15 for bifurcating converging datasets and 30 for linear long, cycle and bifurcating loop datasets. The burn-in period was for each simulation 2. Each network was simulated 32 times.

Simulation of the single-cell RNA-seq experiment

For each dataset we sampled the same number of cells as were present in the reference real dataset, limited to the simulation steps after burn-in. These cells were sampled uniformly across the different steps of the 32 simulations. Next, we used the Splatter package [38] to estimate the different characteristics of a real dataset, such as the distributions of average gene expression, library sizes and dropout probabilities. We used Splatter to simulate the expression levels $\lambda_{i,j}$ of housekeeping genes i (to match the number of genes in the reference dataset) in every cell j. These were combined with the expression levels of the genes simulated within a trajectory. Next, true counts were simulated using $Y'_{i,j} \sim \text{Poisson}(\lambda_{i,j})$. Finally, we simulated dropouts by setting true counts to zero by sampling from a Bernoulli distribution using a dropout probability $\pi^D_{i,j} = \frac{1}{1+e^{-k(\ln(\lambda_{i,j})-x_0)}}$. Both x_0 (the midpoint for the dropout logistic function) and k (the shape of the dropout logistic function) were estimated by Splatter.

This count matrix was then filtered and normalised using the pipeline described below.

Gold standard extraction

Because each cellular simulation follows the trajectory at its own speed, knowing the exact position of a cell within the trajectory topology is not straightforward. Furthermore, the speed at which simulated cells make a decision between two or more alternative paths is highly variable. We therefore first constructed a backbone expression profile for each branch within the trajectory. To do this,

we first defined in which order the expression of the modules is expected to change, and then generated a backbone expression profile in which the expression of these modules increases and decreases smoothly between 0 and 1. We also smoothed the expression in each simulation using a rolling mean with a window of 50 time steps, and then calculated the average module expression along the simulation. We used dynamic time warping, implemented in the dtw R package [100, 101], with an open end to align a simulation to all possible module progressions, and then picked the alignment which minimised the normalised distance between the simulation and the backbone. In case of cyclical trajectory topologies, the number of possible milestones a backbone could progress through was limited to 20.

dyntoy

For more simplistic data generation ("toy" datasets), we created the dyntoy workflow (https://github.com/dynverse/dyntoyhttps://github.com/dynverse/dyntoy) . We created 12 topology generators (described below), and with 10 datasets per generator, this lead to a total of 120 datasets.

We created a set of topology generators, were $B(n, p)$ denotes a binomial distribution, and $U(a, b)$ denotes a uniform distribution:

* Linear and cyclic, with number of milestones $\sim B(10, 0.25)$ * Bifurcating and converging, with four milestones * Binary tree, with number of branching points $\sim U(3, 6)$ * Tree, with number of branching points $\sim U(3, 6)$ and maximal degree $\sim U(3, 6)$

For more complex topologies we first calculated a random number of "modifications" $\sim U(3, 6)$ and a $deg_{max} \sim B(10, 0.25) + 1$. For each type of topology, we defined what kind of modifications are possible: divergences, loops, convergences and divergence-convergence. We then iteratively constructed the topology by uniformly sampling from the set of possible modifications, and adding this modification to the existing topology. For a divergence, we connected an existing milestone to a number of a new milestones. Conversely, for a convergence we connected a number of new nodes to an existing node. For a loop, we connected two existing milestones with a number of milestones in between. Finally for a divergence-convergence we connected an existing milestone to several new milestones which again converged on a new milestone. The number of nodes was sampled from $\sim B(deg_{max} - 3, 0.25) + 2$

* Looping, allowed loop modifications * Diverging-converging, allowed divergence and converging modifications * Diverging with loops, allowed divergence and loop modifications * Multiple looping, allowed looping modifications * Connected, allowed looping, divergence and convergence modifications * Disconnected, number of components sampled from $\sim B(5, 0.25)+2$, for each component we randomly chose a topology from the ones listed above

After generating the topology, we sampled the length of each edge $\sim U(0.5, 1)$. We added regions of delayed commitment to a divergence in a random half of the cases. We then placed the number of cells (same number as from the reference real dataset), on this topology uniformly, based on the length of the edges in the milestone network.

For each gene (same number as from the reference real dataset), we calculated the Kamada-Kawai layout in 2 dimensions, with edge weight equal to the length of the edge. For this gene, we then extracted for each cell a density value using a bivariate normal distribution with $\mu \sim U(x_{min}, x_{min})$ and $\sigma \sim U(x_{min}/10, x_{min}/8)$. We used this density as input for a zero-inflated negative binomial distribution with $\mu\, U(100, 1000) \times density$, $k\, U(\mu/10, \mu/4)$ and pi from the parameters of the reference real dataset, to get the final count values.

This count matrix was then filtered and normalised using the pipeline described below.

PROSSTT

PROSSTT is a recent data simulator [39], which simulates expression using linear mixtures of expression programs and random walks through the trajectory. We used 5 topology generators from dyntoy (linear, bifurcating, multifurcating, binary tree and tree), and simulated for each topology generator 10 datasets using different reference real datasets. However, due to frequent crashes of the tool, only 19 datasets created output and were thus used in the evaluation.

Using the simulate_lineage function, we simulated the lineage expression, with parameters $a \sim U(0.01, 0.1)$, *branch-tol*$_{intra} \sim U(0, 0.9)$ and *branch-tol*$_{inter} \sim U(0, 0.9)$. These parameter distributions were chosen very broad so as to make sure both easy and difficult datasets are simulated. After simulating base gene expression with simulate_base_gene_exp, we used the sample_density function to finally simulate expression values of a number of cells (the same as from the reference real dataset), with $\alpha \sim$ *Lognormal* ($\mu = 0.3$ and $\sigma = 1.5$) and $\beta \sim$ *Lognormal* ($\mu = 2$ and $\sigma = 1.5$). Each of these parameters were centered around the default values of PROSSTT, but with enough variability to ensure a varied set of datasets.

This count matrix was then filtered and normalised using the pipeline described below.

Splatter

Splatter [38] simulates expression values by constructing non-linear paths between different states, each having a distinct expression profile. We used 5 topology generators from dyntoy (linear, bifurcating, multifurcating, binary tree and tree), and simulated for each topology generator 10 datasets using different reference real datasets, leading to a total of 50 datasets.

We used the splatSimulatePaths function from Splatter to simulate datasets, with number of cells and genes equal to those in the reference real dataset, and with parameters *nonlinearProb*, *sigmaFac* and *skew* all sampled from $U(0, 1)$.

3.4.6 Dataset filtering and normalization

We used a standard single-cell RNA-seq preprocessing pipeline that applies parts of the scran and scater Bioconductor packages [102]. The advantages of this pipeline are that it works both with and without spike-ins, and it includes a harsh cell filtering that looks at abnormalities in library sizes, mitochondrial gene expression and the number of genes expressed using median absolute deviations (which we set to 3). We required that a gene was expressed in at least 5% of the cells and that it should have an average expression higher than 0.02. Furthermore, we used the pipeline to select the most highly variable genes, using a false discovery rate of 5% and a biological component higher than 0.5. As a final filter, we removed both all-zero genes and cells until convergence.

3.4.7 Benchmark metrics

The importance of using multiple metrics to compare complex models has been stated repeatedly [90]. Furthermore, a trajectory is a model with multiple layers of complexity, which calls for several metrics each assessing a different layer. We therefore defined several possible metrics for comparing trajectories, each investigating different layers. These are all discussed in Supplementary Note 1 along with examples and robustness analyses when appropriate.

Next, we created a set of rules to which we think a good trajectory metric should conform, and tested this empirically for each metric by comparing scores before and after perturbing a dataset (Supplementary Note 1). Based on this analysis, we chose four metrics for the evaluation, each assessing a different aspect of the trajectory: (1) the HIM measures the topological similarity; (2) the F1branches

compares the branch assignment; (3) the cordist assesses the similarity in pairwise cell–cell distances and thus the cellular positions; and (4) the wcorfeatures looks at whether similar important features (genes) are found in both the reference dataset and the prediction.

The Hamming–Ipsen–Mikhailov metric

The HIM metric [103] uses the two weighted adjacency matrices of the milestone networks as input (weighted by edge length). It is a linear combination of the normalized Hamming distance, which gives an indication of the differences in edge lengths, and the normalized Ipsen–Mikhailov distance, which assesses the similarity in degree distributions. The latter has a parameter γ, which was fixed at 0.1 to make the scores comparable between datasets. We illustrate the metric and discuss alternatives in Supplementary Note 1.

The F1 between branch assignments

To compare branch assignment, we used an F1 score, also used used for comparing biclustering methods [88]. To calculate this metric, we first calculated the similarity of all pairs of branches between the two trajectories using the Jaccard similarity. Next, we defined the 'Recovery' (respectively 'Relevance') as the average maximal similarity of all branches in the reference dataset (respectively prediction). The F1branches was then defined as the harmonic mean between Recovery and Relevance. We illustrate this metric further in Supplementary Note 1.

Correlation between geodesic distances

When the position of a cell is the same in both the reference and the prediction, its relative distances to all other cells in the trajectory should also be the same. This observation is the basis for the cordist metric. To calculate the cordist, we first sampled 100 waypoint cells in both the prediction and the reference dataset, using stratified sampling between the different milestones, edges and regions of delayed commitment, weighted by the number of cells in each collection. We then calculated the geodesic distances between the union of waypoint cells from both datasets and all other cells. The calculation of the geodesic distance depended on the location of the two cells within the trajectory, further discussed in Supplementary Note 1, and was weighted by the length of the edge in the milestone network. Finally, the cordist was defined as the Spearman rank correlation between the distances of both datasets. We illustrate the metric and assess the effect of the number of waypoint cells in Supplementary Note 1.

The correlation between important features

The wcorfeatures assesses whether the same differentially expressed features are found using the predicted trajectory as in the known trajectory. To calculate this metric, we used Random Forest regression (implemented in the R ranger package [104]), to predict expression values of each gene, based on the geodesic distances of a cell to each milestone. We then extracted feature importance values for each feature and calculated the similarity of the feature importances using a weighted Pearson correlation, weighted by the feature importance in the reference dataset to give more weight to large differences. As hyperparameters we set the number of trees to 10,000 and the number of features on which to split to 1% of all available features. We illustrate this metric and assess the effect of its hyperparameters in Supplementary Note 1.

Score aggregation

To rank methods, we needed to aggregate the different scores on two levels: across datasets and across different metrics. This aggregation strategy is explained in more detail in Supplementary Note 1.

To ensure that easy and difficult datasets have equal influence on the final score, we first normalized the scores on each dataset across the different methods. We shifted and scaled the scores to σ = 1 and μ = 0, and then applied the unit probability density function of a normal distribution on these values to get the scores back into the [0,1] range.

Since there is a bias in dataset source and trajectory type (for example, there are many more linear datasets), we aggregated the scores per method and dataset in multiple steps. We first aggregated the datasets with the same dataset source and trajectory type using an arithmetic mean of their scores. Next, the scores were averaged over different dataset sources, using an arithmetic mean that was weighted based on how much the synthetic and silver scores correlated with the real gold scores. Finally, the scores were aggregated over the different trajectory types again using an arithmetic mean.

Finally, to get an overall benchmarking score, we aggregated the different metrics using a geometric mean.

3.4.8 Method execution

Each execution of a method on a dataset was performed in a separate task as part of a gridengine job. Each task was allocated one CPU core of an Intel(R) Xeon(R) CPU E5-2665 at 2.40 GHz, and one R session was started for each task. During the execution of a method on a dataset, if the time limit (>1 h) or memory limit (16 GB) was exceeded, or an error was produced, a zero score was returned for that execution.

3.4.9 Complementarity

To assess the complementarity between different methods, we first calculated for every method and dataset whether the overall score was equal to or higher than 95% of the best overall score for that particular dataset. We then calculated for every method the weighted percentage of datasets that fulfilled this rule, weighted similarly as in the benchmark aggregation, and chose the best method. We iteratively added new methods until all methods were selected. For this analysis, we did not include any methods that require any strong prior information and only included methods that could detect the trajectory types present in at least one of the datasets.

3.4.10 Scalability

To assess the scalability of each method, we started from five real datasets, selected using the centers from a k-medoids as discussed before. We up- and downscaled these datasets between 10 and 100,000 cells and 10 and 100,000 features, while never going higher than 1,000,000 values in total. To generate new cells or features, we first generated a 10-nearest-neighbor graph of both the cells and features from the expression space. For every new cell or feature, we used a linear combination of one to three existing cells or features, where each cell or feature was given a weight sampled from a uniform distribution between 0 and 1.

We ran each method on each dataset for maximally 1 h and gave each process 10 GB of memory. To determine the running time of each method, we started the timer right after data loading and the loading of any packages, and stopped the clock before postprocessing and saving of the output. Pre- and postprocessing steps specific to a method, such as dimensionality reduction and gene filtering,

were included in the time. To estimate the maximal memory usage, we used the max_vmem value from the qacct command provided by a gridengine cluster. We acknowledge, however, that these memory estimates are very noisy and the averages provided in this study are therefore only rough estimates.

The relationship between the dimensions of a dataset and the running time or maximal memory usage was modeled using shape constrained additive models [69], with $log_{10}|cells|$ and $log_{10}|features|$ as predictor variables, and fitted this model using the scam function as implemented in the R scam package, with $log_{10}time$ (or $log_{10}memory$) as outcome.

To classify the time complexity of each method with respect to the number of cells, we predicted the running time at 10,000 features with increasing number of cells from 100 to 100,000, with steps of 100. We trained a generalized linear model with the following function: $y \approxeq \log x + \sqrt{x} + x + x^2 + x^3$ with y as running time and x as the number of cells or features. The time complexity of a method was then classified using the weights w from this model:

$$\begin{cases} \text{superquadratic} & \text{if } w_{x^3} > 0.25, \\ \text{quadratic} & \text{if } w_{x^2} > 0.25, \\ \text{linear} & \text{if } w_x > 0.25, \\ \text{sublinear} & \text{if } w_{\log(x)} > 0.25 \text{ or } w_{\text{sqrt}(x)} > 0.25, \\ \text{case with highest weight} & \text{else.} \end{cases}$$

This process was repeated for classifying the time complexity with respect to the number of features, and the memory complexity both with respect to the number of cells and features.

3.4.11 Stability

In the ideal case, a method should produce a similar trajectory, even when the input data is slightly different. However, running the method multiple times on the same input data would not be the ideal approach to assess its stability, given that a lot of tools are artificially deterministic by internally resetting the pseudorandom number generator (for example, using the 'set.seed' function in R or the 'random.seed' function in numpy). To assess the stability of each method, we therefore selected a number of datasets, which consisted of 25% of the datasets accounting for 15% of the total runtime, chosen such that after aggregation the overall scores still has > 0.99 correlation with the original overall ranking. We subsampled each dataset 10 times with 95% of the original cells and 95% of the original features. We ran every method on each of the bootstraps, and assessed the stability by calculating the benchmarking scores between each pair of subsequent models (run i is compared to run $i + 1$). For the cordist and F1branches, we only used the intersection between the cells of two datasets, while the intersection of the features was used for the wcorfeatures.

3.4.12 Usability

We created a transparent scoring scheme to quantify the usability of each method based on several existing tool quality and programming guidelines in the literature and online (Table 3.1). The main goal of this quality control is to stimulate the improvement of current methods, and the development of user- and developer-friendly new methods. The quality control assessed six categories, each looking at several aspects, which are further divided into individual items. The availability category checks whether the method is easily available, whether the code and dependencies can be easily installed, and how the method can be used. The code quality assesses the quality of the code both from a user

perspective (function naming, dummy proofing and availability of plotting functions) and a developer perspective (consistent style and code duplication). The code assurance category is frequently overlooked, and checks for code testing, continuous integration [80] and an active support system. The documentation category checks the quality of the documentation, both externally (tutorials and function documentation) and internally (inline documentation). The behavior category assesses the ease by which the method can be run, by looking for unexpected output files and messages, prior information and how easy the trajectory model can be extracted from the output. Finally, we also assessed certain aspects of the study in which the method was proposed, such as publication in a peer-reviewed journal, the number of datasets in which the usefulness of the method was shown and the scope of method evaluation in the paper.

Each quality aspect received a weight depending on how frequently it was found in several papers and online sources that discuss tool quality (Table 3.1). This was to make sure that more important aspects, such as the open source availability of the method, outweighed other less important aspects, such as the availability of a graphical user interface. For each aspect, we also assigned a weight to the individual questions being investigated (Table 3.1). For calculating the final score, we weighed each of the six categories equally.

3.4.13 Guidelines

For each set of outcomes in the guidelines figure, we selected one to four methods, by first filtering the methods on those that can detect all required trajectory types, and ordering the methods according to their average accuracy score on datasets containing these trajectory types (aggregated according to the scheme presented in the section Accuracy).

We used the same approach for selecting the best set of methods in the guidelines app (http://guidelines.dynverse.c developed using the R shiny package. This app will also filter the methods, among other things, depending on the predicted running time and memory requirements, the prior information available and the preferred execution environment (using the dynmethods package or standalone).

3.4.14 Reporting Summary

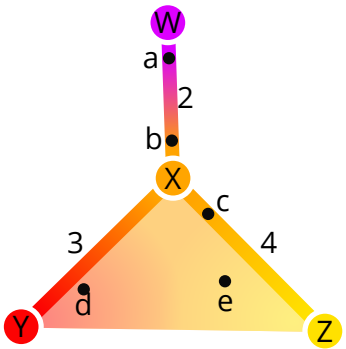
Further information on research design is available in the Nature Research Reporting Summary, available at https://www.nature.com/articles/s41587-019-0071-9#MOESM2

3.5 Supplementary Note 1: Metrics to compare two trajectories

A trajectory, as defined in our evaluation, is a model with multiple abstractions. The top abstraction is the topology which contains information about the paths each cell can take from their starting point. Deeper abstractions involve the mapping of each cell to a particular branch within this network, and the position (or ordering) of each cells within these branches. Internally, the topology is represented by the milestone network and regions of delayed commitment, the branch assignment and cellular positions are represented by the milestone percentages (Figure 3.10).

Given the multilayered complexity of a trajectory model, it is not trivial to compare the similarity of two trajectory models using only one metric. We therefore sought to use different comparison metrics, each serving a different purpose:

- **Specific metrics** investigate one particular aspect of the trajectory. Such metrics make it possible to find particular weak points for methods, e.g. that a method is very good at ordering but does not frequently find the correct topology. Moreover, having multiple individual metrics allow



Milestone network

from	to	length
W	X	2
X	Y	3
X	Z	4

Regions of delayed commitment

region	to	is_begin
XYZ	X	TRUE
XYZ	Y	FALSE
XYZ	Z	FALSE

Branch assignment

Represented by the milestone percentages

cell	milestone	percentage
a	W	0.9
a	X	0.1
b	W	0.2
b	X	0.8
c	X	0.8
c	Z	0.2
d	X	0.2
d	Y	0.7
d	Z	0.1
e	X	0.3
e	Y	0.2
e	Z	0.5

Cell positions

Figure 3.10: An example trajectory that will be used throughout this section. It contains contains four milestones (W to Z) and five cells (a to e).

personalised rankings of methods, for example for users which are primarily interested in using the method correct topology.

- **Application metrics** focus on the quality of a downstream analysis using the trajectory. For example, it measures whether the trajectory can be used to find accurate differentially expressed genes.

- **Overall metrics** should capture all the different abstractions, in other words such metrics measure whether the resulting trajectory has a good topology, that the cells belong to similar branches *and* that they are ordered correctly.

Here, we first describe and illustrate several possible specific, application and overall metrics. Next, we test these metrics on several test cases, to make sure they robustly identify "wrong" trajectory predictions.

All metrics described here were implemented within the dyneval R package (https://github.com/dynverse/dyneval).

3.5.1 Metric characterisation and testing

isomorphic, *edgeflip* and *HIM*: Edit distance between two trajectory topologies

We used three different scores to assess the similarity in the topology between two trajectories, irregardless of where the cells were positioned.

For all three scores, we first simplified the topology of the trajectory to make both graph structures comparable:

- As we are only interested in the main structure of the topology without start or end, the graph was made undirected.
- All milestones with degree 2 were removed. For example in the topology A ⇒ B ⇒ C ⇒ D, C ⇒ D, the B milestone was removed
- A linear topology was converted to A ⇒ B ⇒ C
- A cyclical topology such as A ⇒ B ⇒ C ⇒ D or A ⇒ B ⇒ A were all simplified to A ⇒ B ⇒ C ⇒ A
- Duplicated edges such as A ⇒ B, A ⇒ B were decoupled to A ⇒ B, A ⇒ C ⇒ B

The *isomorphic* score returns 1 if two graphs are isomorphic, and 0 if they were not. For this, we used the used the BLISS algorithm [105], as implemented in the R *igraph* package.

The *edgeflip* score was defined as the minimal number of edges which should be added or removed to convert one network into the other, divided by the total number of edges in both networks. This problem is equivalent to the maximum common edge subgraph problem, a known NP-hard problem without a scalable solution [106]. We implemented a branch and bound approach for this problem, using several heuristics to speed up the search:

- First check all possible edge additions and removals corresponding to the number of different edges between the two graphs.
- For each possible solution, first check whether:
  1. The maximal degree is the same
  2. The minimal degree is the same
  3. All degrees are the same after sorting
- Only then check if the two graphs are isomorphic as described earlier.
- If no solution is found, check all possible solutions with two extra edge additions/removals.

The *HIM* metric (Hamming-Ipsen-Mikhailov distance) [103] which was adopted from the R nettools package (https://github.com/filosi/nettools). It uses an adjacency matrix which was weighted according to the lengths of each edges within the milestone network. Conceptually, *HIM* is a linear combination of:

- The normalised Hamming distance [107], which calculates the distance between two graphs by matching individual edges in the adjacency matrix, but disregards overall structural similarity.
- The normalised Ipsen-Mikhailov distance [108], which calculates the overall distance of two graphs based on matches between its degree and adjacency matrix, while disregarding local structural similarities. It requires a γ parameter, which is usually estimated based on the number of nodes in the graph, but which we fixed at 0.1 so as to make the score comparable across different graph sizes.

We compared the three scores on several common topologies (Figure 3.11a). While conceptually very different, the *edgeflip* and *HIM* still produce similar scores (Figure 3.11b). The *HIM* tends to punish the detection of cycles, while the *edgeflip* is more harsh for differences in the number of bifurcations (Figure 3.11b). The main difference however is that the *HIM* takes into account edge lengths when comparing two trajectories, as illustrated in (Figure 3.11c). Short "extra" edges in the topology are less punished by the *HIM* than by the *edgeflip*.

To summarise, the different topology based scores are useful for different scenarios:

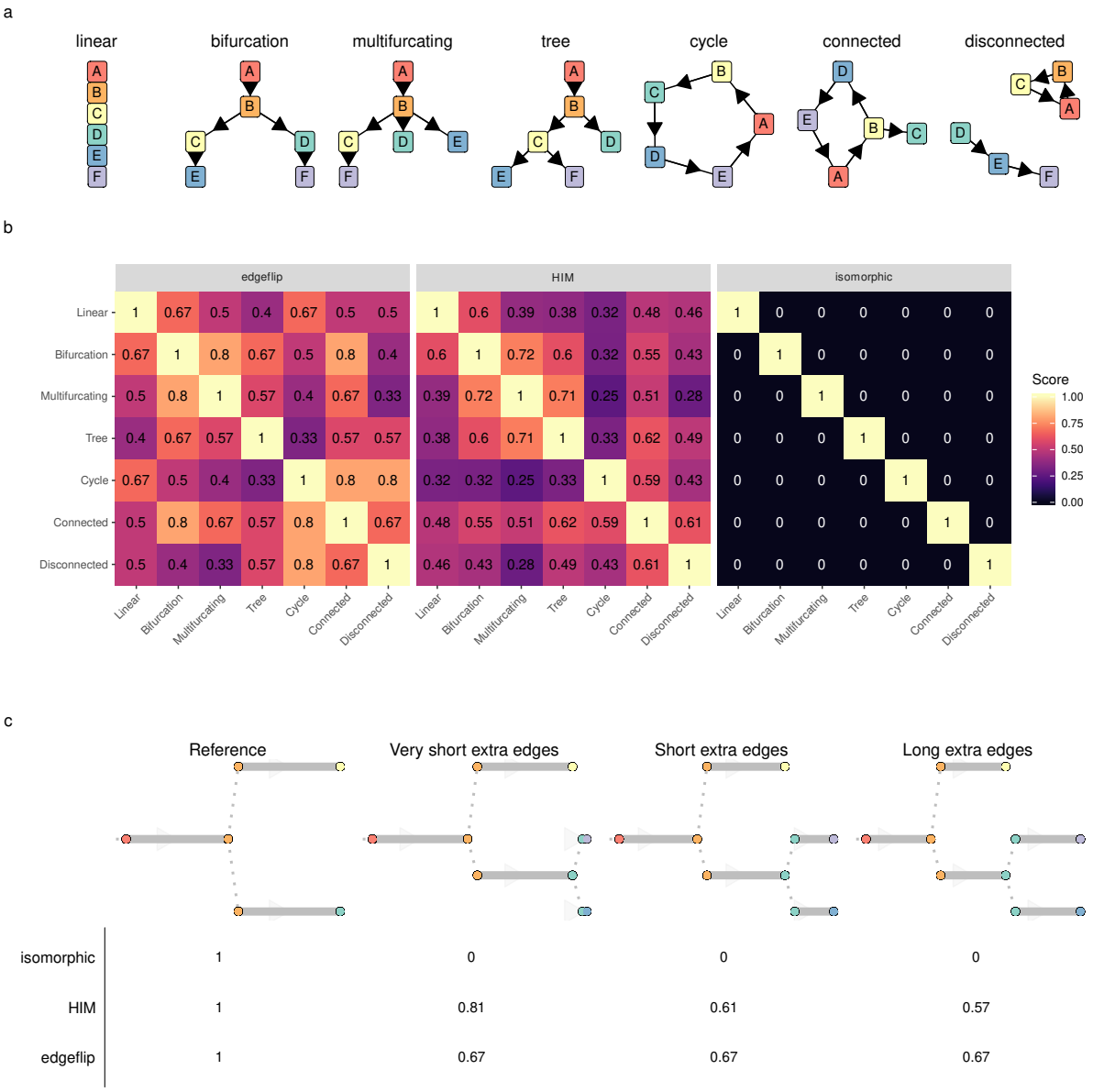


Figure 3.11: Showcase of three metrics to evaluate topologies: *isomorphic*, *edgeflip* and *HIM* (a) The used topologies. **(b)** The scores when comparing each pair of trajectory types. (c) Four datasets in which aan extra edge is added and made progressively longer. This shows how the HIM can take into account edge lengths.

- If the two trajectories should only be compared when the topology is exactly the same, the *isomorphic* should be used.
- If it is important that the topologies are similar, but not necessarily isomorphic, the *edgeflip* is most appropriate.
- If the topologies should be similar, but shorter edges should not be punished as hard as longer edges, the *HIM* is most appropriate.

$F1_{branches}$ and $F1_{milestones}$: Comparing how well the cells are clustered in the trajectory

Perhaps one of the simplest ways to calculate the similarity between the cellular positions of two topologies is by mapping each cell to its closest milestone *or* branch 3.12. These clusters of cells can then be compared using one of the many external cluster evaluation measures [88]. When selecting a cluster evaluation metric, we had two main conditions:

- Because we allow methods to filter cells in the trajectory, the metric should be able to handle "non-exhaustive assignment", where some cells are not assigned to any cluster. - The metric should give each cluster equal weight, so that rare cell stages are equally important as large stages.

The *F1* score between the *Recovery* and *Relevance* is a metric which conforms to both these conditions. This metric will map two clustersets by using their shared members based on the *Jaccard* similarity. It then calculates the *Recovery* as the average maximal *Jaccard* for every cluster in the first set of clusters (in our case the reference trajectory). Conversely, the *Relevance* is calculated based on the average maximal similarity in the second set of clusters (in our case the prediction). Both the *Recovery* and *Relevance* are then given equal weight in a harmonic mean (*F1*). Formally, if C and C' are two cell clusters:

$$\begin{aligned}
\mathit{Jaccard}(c, c') &= \frac{|c \cap c'|}{|c \cup c'|} \\
\mathit{Recovery} &= \frac{1}{|C|} \sum_{c \in C} \max_{c' \in C'} \mathit{Jaccard(c, c')} \\
\mathit{Relevance} &= \frac{1}{|C'|} \sum_{c' \in C'} \max_{c \in C} \mathit{Jaccard(c, c')} \\
\mathit{F1} &= \frac{2}{\frac{1}{\mathit{Recovery}} + \frac{1}{\mathit{Relevance}}}
\end{aligned}$$

cor_{dist}: Correlation between geodesic distances

When the position of a cell is the same in both the reference and the prediction, its *relative* distances to all other cells in the trajectory should also be the same. This observation is the basis for the cor_{dist} metric.

The geodesic distance is the distance a cell has to go through the trajectory space to get from one position to another. The way this distance is calculated depends on how two cells are positioned, showcased by an example in Figure 3.13:

- **Both cells are on the same edge in the milestone network.** In this case, the geodesic distance is defined as the product of the difference in milestone percentages and the length of their shared edge. For cells a and b in the example, $d(a, b)$ is equal to $1 \times (0.9 - 0.2) = 0.7$.

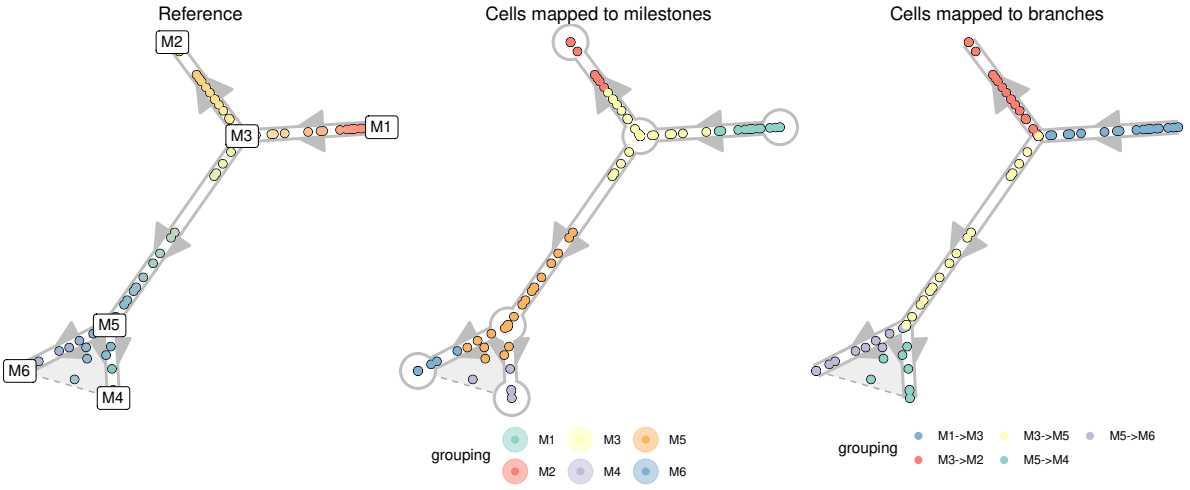
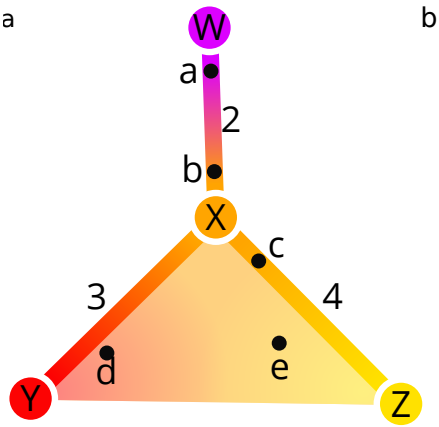


Figure 3.12: Mapping cells to their closest milestone or branch for the calculation of the $F1_{milestones}$ and $F1_{branches}$. To calculate the $F1_{milestones}$, cells are mapped towards the nearest milestone, i.e. the milestone with the highest milestone percentage. For the $F1_{branches}$, the cells are mapped to the closest edge.

a



b

Milestone network

from	to	length
W	X	2
X	Y	3
X	Z	4

Region of delayed commitment

region	to	is_begin
XYZ	X	TRUE
XYZ	Y	FALSE
XYZ	Z	FALSE

Milestone percentages

cell	milestone	percentage
a	W	0.9
a	X	0.1
b	W	0.2
b	X	0.8
c	X	0.8
c	Z	0.2
d	X	0.2
d	Y	0.7
d	Z	0.1
e	X	0.3
e	Y	0.2
e	Z	0.5

c

d(a, b) = 2 × (0.9 - 0.2) = 1.4
d(a, c) = 2 × 0.9 + 4 × 0.2 = 2.6
d(b, c) = 2 × 0.2 + 4 × 0.2 = 1.2
d(a, d) = 2 × 0.9 + 3 × 0.7 + 4 × 0.1 = 4.3
d(b, d) = 2 × 0.2 + 3 × 0.7 + 4 × 0.1 = 2.9
d(c, d) = 3 × (0.7 - 0) + 4 × (0.2 - 0.1) = 2.5
d(a, e) = 2 × 0.9 + 3 × 0.2 + 4 × 0.5 = 4.4
d(b, e) = 2 × 0.2 + 3 × 0.2 + 4 × 0.5 = 3.0
d(c, e) = 3 × (0.2 - 0) + 4 × (0.5 - 0.2) = 1.8
d(d, e) = 3 × (0.7 - 0.2) + 4 × (0.5 - 0.1) = 3.1

d

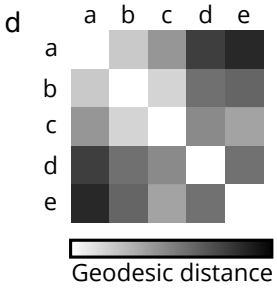


Figure 3.13: The calculation of geodesic distances on a small example trajectory. a) A toy example containing four milestones (W to Z) and five cells (a to e). **b)** The corresponding milestone network, milestone percentages and regions of delayed commitment, when the toy trajectory is converted to the common trajectory model. **c)** The calculations made for calculating the pairwise geodesic distances. **d)** A heatmap representation of the pairwise geodesic distances.

- **Cells reside on different edges in the milestone network.** First, the distance of the cell to all its nearby milestones is calculated, based on its percentage within the edge and the length of the edge. These distances in combination with the milestone network are used to calculate the shortest path distance between the two cells. For cells a and c in the example, $d(a, X) = 1 \times 0.9$ and $d(c, X) = 3 \times 0.2$, and therefore $d(a, c) = 1 \times 0.9 + 3 \times 0.2$.

The geodesic distance can be easily extended towards cells within regions of delayed commitment. When both cells are part of the same region of delayed commitment, the geodesic distance was defined as the manhattan distances between the milestone percentages weighted by the lengths from the milestone network. For cells d and e in the example, $d(d, e)$ is equal to $0 \times (0.3 - 0.2) + 2 \times (0.7 - 0.2) + 3 \times (0.4 - 0.1) = 1.9$. The distance between two cells where only one is part of a region of delayed commitment is calculated similarly to the previous paragraph, by first calculating the distance between the cells and their neighbouring milestones first, then calculating the shortest path distances between the two.

Calculating the pairwise distances between cells scales quadratically with the number of cells, and would therefore not be scaleable for large datasets. For this reason, a set of waypoint cells are defined *a priori*, and only the distances between the waypoint cells and all other cells is calculated, in order to calculate the correlation of geodesic distances of two trajectories (Figure 3.14a). These cell waypoints are determined by viewing each milestone, edge and region of delayed commitment as a collection of cells. We do stratified sampling from each collection of cells by weighing them by the total number of cells within that collection. For calculating the cor_{dist} between two trajectories, the distances between all cells and the union of both waypoint sets is computed.

To select the number of cell waypoints, we need to find a trade-off between the accuracy versus the time to calculate cor_{dist}. To select an optimal number of cell waypoints, we used the synthetic dataset with the most complex topology, and determined the cor_{dist} at different levels of both cell shuffling and number of cell waypoints (Figure 3.14a). We found that using cell waypoints does not induce a systematic bias in the cor_{dist}, and that its variability was relatively minimal when compared to the variability between different levels of cell shuffling when using 100 or more cell waypoints.

Although the cor_{dist}'s main characteristic is that it looks at the positions of the cells, other features of the trajectory are also (partly) captured. To illustrate this, we used the geodesic distances themselves as input for dimensionality reduction (Figure 3.15) with varying topologies. This reduced space captures the original trajectory structure quite well, including the overall topology and branch lengths.

$NMSE_{rf}$ and $NMSE_{lm}$: Using the positions of the cells within one trajectory to predict the cellular positions in the other trajectory

An alternative approach to detect whether the positions of cells are similar between two trajectories, is to use the positions of one trajectory to predict the positions within the other trajectory. If the cells are at similar positions in the trajectory (relative to its nearby cells), the prediction error should be low.

Specifically, we implemented two metrics which predict the milestone percentages from the reference by using the predicted milestone percentages as features (Figure 3.16). We did this with two regression methods, linear regression (*lm*, using the R lm function) and Random Forest (*rf*, implemented in the *ranger* package [104]). In both cases, the accuracy of the prediction was measured using the Mean Squared error (MSE), in the case of Random forest we used the out-of-bag mean-squared error. Next, we calculated MSE_{worst} equal to the MSE when predicting all milestone percentages as the average. We used this to calculate the normalised mean squared error as $NMSE = 1 - \frac{MSE}{MSE_{worst}}$. We

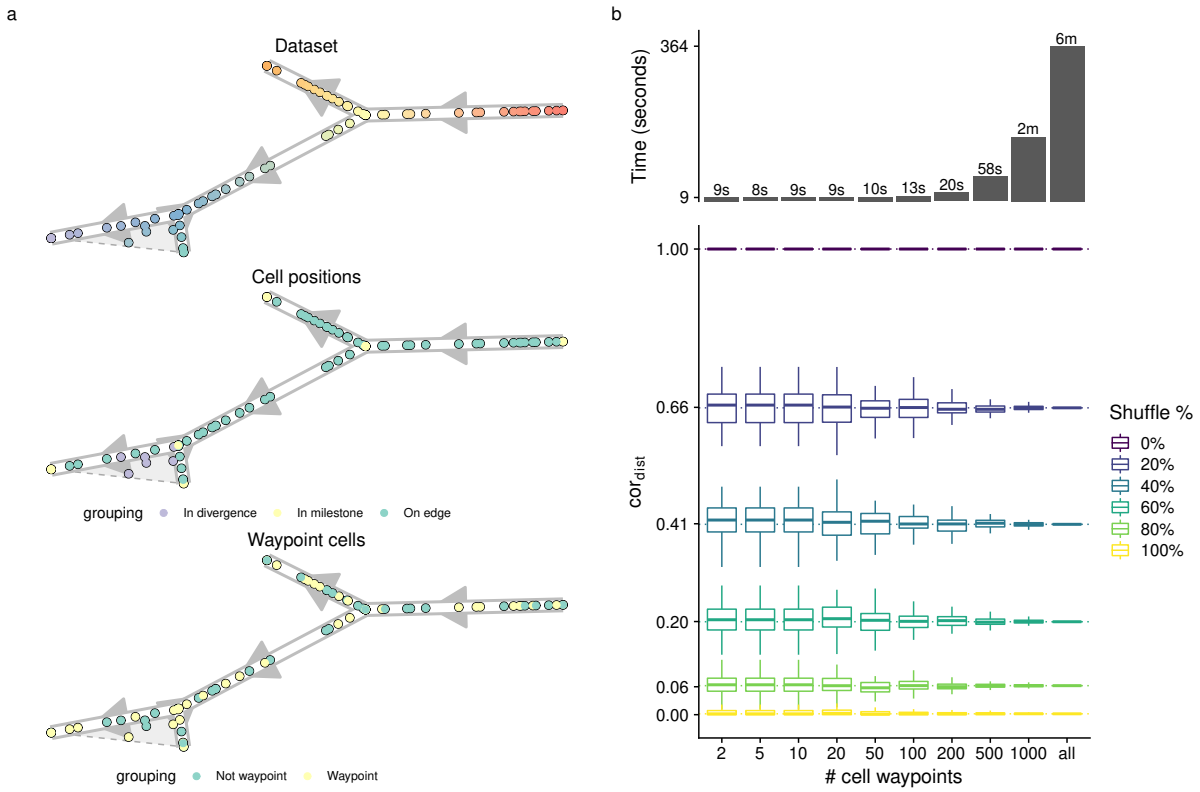


Figure 3.14: Determination of cell waypoints a) Illustration of the stratified cell sampling using an example dataset (top). Each milestone, edge between two milestones and region of delayed commitment is seen as a collection of cells (middle), and the number of waypoints (100 in this case) are divided over each of these collection of cells (bottom). **b)** Accuracy versus time to calculate cor_{dist}. Shown are distributions over 100 random waypoint samples. The upper whisker of the boxplot extends from the hinge (75% percentile) to the largest value, no further than 1.5× the IQR of the hinge. The lower whisker extends from the hinge (25% percentile) to the smallest value, at most 1.5× the IQR of the hinge.

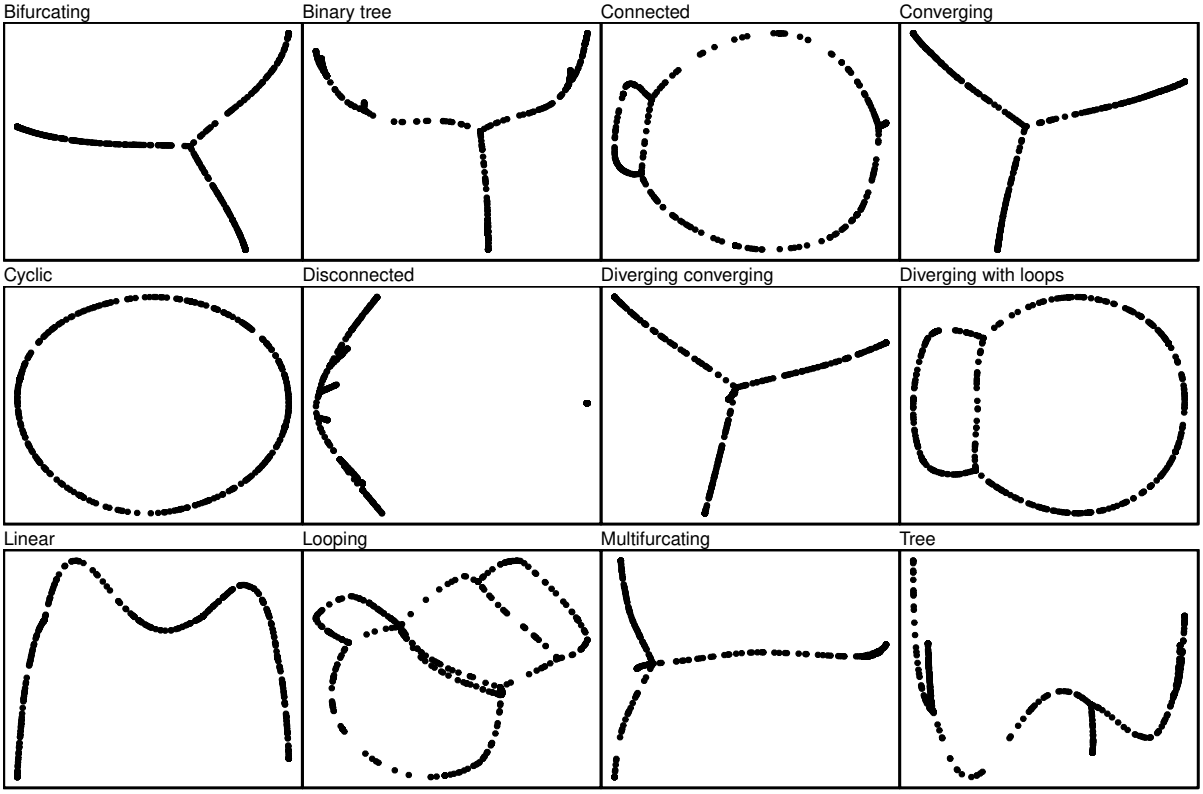


Figure 3.15: Determination of cell waypoints. We generated different toy trajectory datasets with varying topologies and calculated the geodesic distances between all cells within the trajectory. We then used these distances as input for classical multidimensional scaling. This shows that the geodesic distances do not only contain information regarding the cell's positions, but also information on the lengths and wiring of the topology.

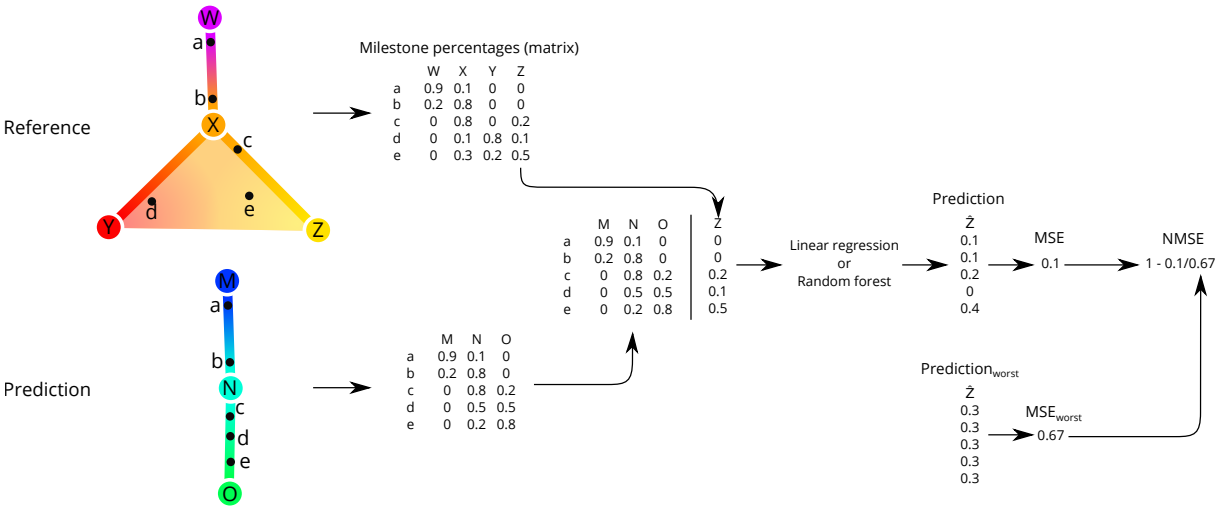


Figure 3.16: The calculation of $NMSE_{lm}$ distances on a small example trajectory. The milestone percentages of the reference are predicted based on the milestone percentages of the prediction, using regression models such as linear regression or random forests. The predicted trajectory is then scored by comparing the mean-squared error (MSE) of this regression model with the baseline MSE where the prediction is the average milestone percentage.

created a regression model for every milestone in the gold standard, and averaged the $NMSE$ values to finally obtain the $NMSE_{rf}$ and $NMSE_{lm}$ scores.

$cor_{features}$ and $wcor_{features}$: The accuracy of dynamical differentially expressed features/genes.

Although most metrics described above already assess some aspects directly relevant to the user, such as whether the method is good at finding the right topology, these metrics do not assess the quality of downstream analyses and hypotheses which can be generated from these models.

Perhaps the main advantage of studying cellular dynamic processes using single-cell -omics data is that the dynamics of gene expression can be studied for the whole transcriptome. This can be used to construct other models such as dynamic regulatory networks and gene expression modules. Such analyses rely on a "good-enough" cellular ordering, so that it can be used to identify dynamical differentially expressed genes.

To calculate the $cor_{features}$ we used Random forest regression to rank all the features according to their importance in predicting the positions of cells in the trajectory. More specifically, we first calculated the geodesic distances for each cell to all milestones in the trajectory. Next, we trained a Random Forest regression model (implemented in the R *ranger* package [104], https://github.com/imbs-hl/ranger) to predict these distances for each milestone, based on the expression of genes within each cell. We then extracted feature importances using the Mean Decrease in Impurity (importance = 'impurity' parameter of the ranger function), as illustrated in Figure 3.17. The overall importance of a feature (gene) was then equal to the mean importance over all milestones. Finally, we compared the two rankings by calculating the Pearson correlation, with values between -1 and 0 clipped to 0.

Random forest regression has two main hyperparameters. The number of trees to be fitted (num_tree parameter) was fixed to 10000 to provide accurate and stable estimates of the feature importance (Figure 3.18. The number of features on which can be split (mtry parameter) was set to 1% of all available features (instead of the default square-root of the number of features), as to make sure that predictive but highly correlated features, omnipresent in transcriptomics data, are not suppressed in the ranking.

For most datasets, only a limited number of features will be differentially expressed in the trajectory. For example, in the dataset used in Figure 3.18 only the top 10%-20% show a clear pattern of differential expression. The correlation will weight each of these features equally, and will therefore give

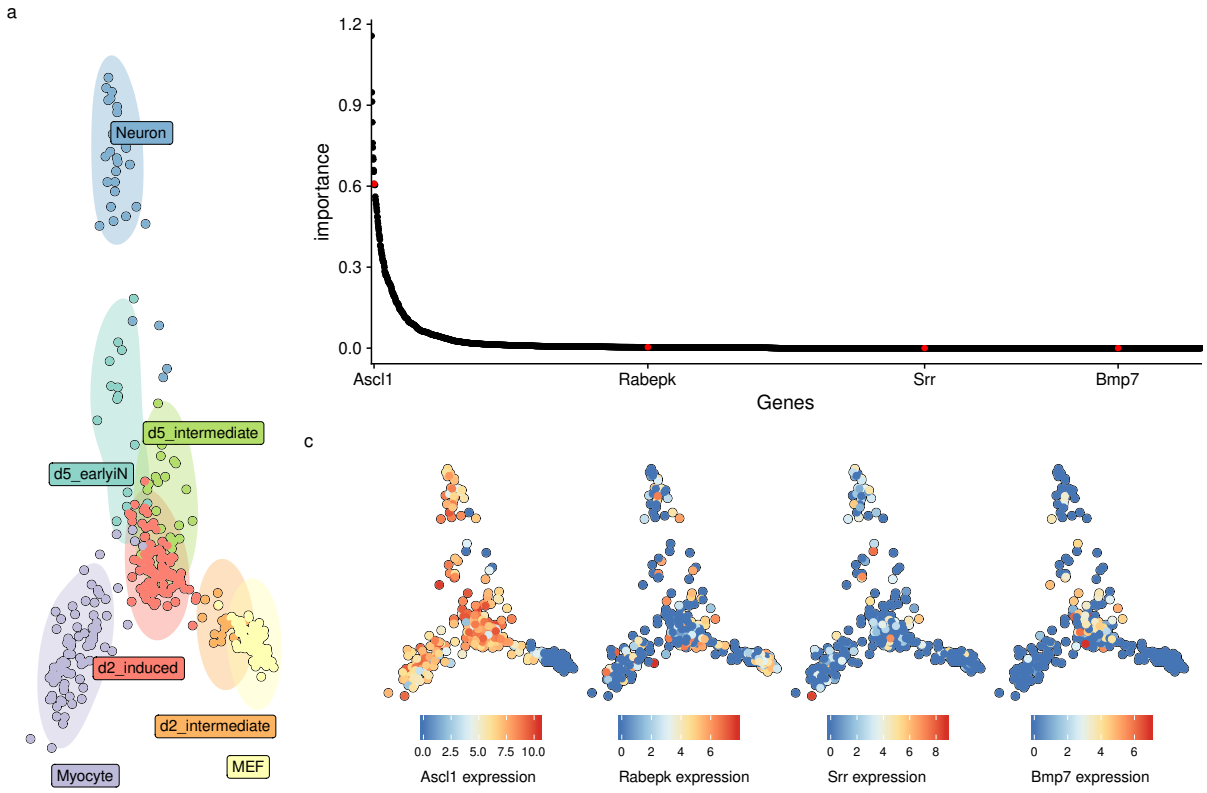


Figure 3.17: An illustration of ranking features based on their importance in a trajectory. (a) A MDS dimensionality reudction of a real dataset in which mouse embryonic fibroblasts (MEF) differentiate into Neurons and Myocytes. (b) The ranking of feature importances from high to low. The majority of features have a very low importance. (c) Some examples, which were also highlighted in b. Higher features in the ranking are clearly specific to certain parts of the trajectory, while features lower on the ranking have a more dispersed expression pattern.

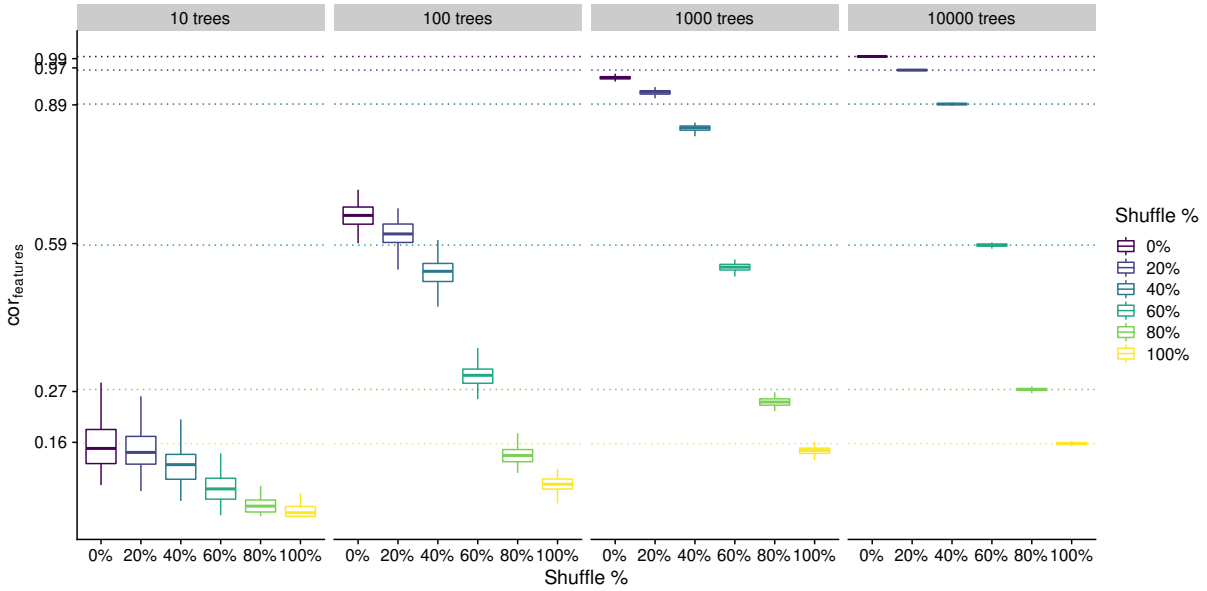


Figure 3.18: Effect of the number of trees parameter on the accuracy and variability of the $cor_{features}$. We used the dataset from Figure 3.17 and calculated the $cor_{features}$ after shuffling a percentage of cells.

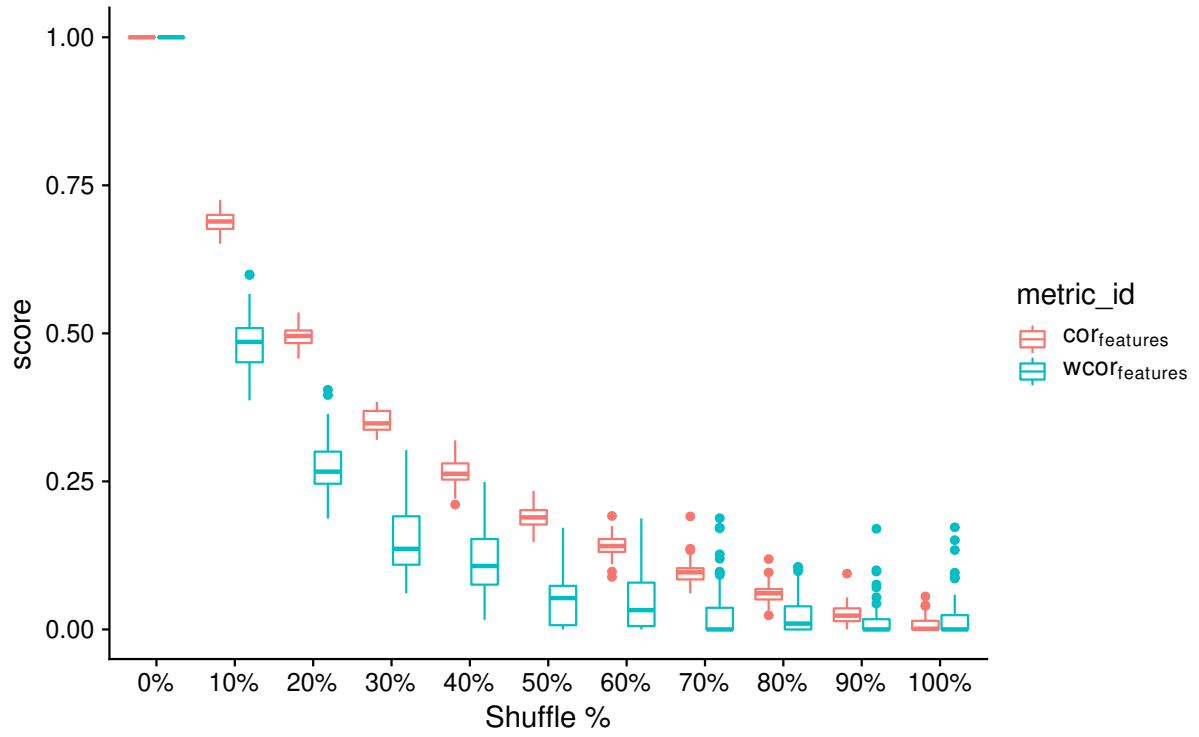


Figure 3.19: Effect of weighting the features based on their feature importance in the reference. We used the same dataset as in Figure 3.17, and calculated the $cor_{features}$ after shuffling a percentage of cells.

more weight to the bottom, irrelevant features. To prioritise the top differentially expressed features, we also implemented the $wcor_{features}$, which will weight the correlation using the feature importance scores in the reference so that the top features have relatively more impact on the score (Figure 3.19).

3.5.2 Metric conformity

Although most metrics described in the previous section make sense intuitively, this does not necessarily mean that these metrics are robust and will generate reasonable results when used for benchmarking. This is because different methods and datasets will all lead to a varied set of trajectory models:

- Real datasets have all cells grouped onto milestones
- Some methods place all cells in a region of delayed commitment, others never generate a region of delayed commitment
- Some methods always return a linear trajectory, even if a bifurcation is present in the data
- Some methods filter cells

A good metric, especially a good overall metric, should work in all these circumstances. To test this, we designed a set of rules to which a good metric should conform, and assessed empirically whether a metric conforms to these rules.

We generated a panel of toy datasets (using our *dyntoy* package, https://github.com/dynverse/dyntoy) with all possible combinations of:

- # cells: 10, 20, 50, 100, 200, 500

Table 3.2: Overview of whether a particular metric conforms to a particular rule

name	cor_{dist}	$NMSE_{rf}$	$NMSE_{lm}$	edgeflip	HIM	isomorphic	$cor_{features}$	$wcor_{features}$	$F1_{branches}$	$F1_{milestones}$	$mean_{geometric}$
Same score on identity	✓	✗	✓	✓	✓	✓	✓	✗	✓	✓	✓
Local cell shuffling	✓	✓	✓	✗	✗	✗	✓	✓	✗	✓	✓
Edge shuffling	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓	✓
Local and global cell shuffling	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓	✓
Changing positions locally and/or globally	✓	✓	✓	✗	✗	✗	✓	✓	✗	✗	✓
Cell filtering	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓	✓
Removing divergence regions	✓	✓	✓	✗	✗	✗	✓	✓	✗	✓	✓
Move cells to start milestone	✓	✓	✓	✗	✗	✗	✓	✓	✗	✓	✓
Move cells to closest milestone	✓	✓	✓	✗	✗	✗	✓	✓	✗	✓	✓
Length shuffling	✓	✗	✓	✗	✓	✗	✗	✗	✗	✓	✓
Cells into small subedges	✗	✓	✗	✓	✓	✓	✓	✗	✓	✓	✓
New leaf edges	✓	✓	✗	✓	✓	✓	✗	✗	✓	✓	✓
New connecting edges	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓
Changing topology and cell position	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓
Bifurcation merging	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓
Bifurcation merging and changing cell positions	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓	✓
Bifurcation concatentation	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓
Cycle breaking	✓	✗	✓	✓	✓	✓	✓	✓	✗	✓	✓
Linear joining	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓
Linear splitting	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Change of topology	✓	✗	✗	✓	✓	✓	✓	✓	✗	✓	✓
Cells on milestones vs edges	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

- # features: 200
- topologies: linear, bifurcation, multifurcating, tree, cycle, connected graph and disconnected graph
- Whether cells are placed on the milestones (as in real data) or on the edges/regions of delayed commitment between the milestones (as in synthetic data)

We then perturbed the trajectories in these datasets in certain ways, and tested whether the scores follow an expected pattern. An overview of the conformity of every metric is first given in Table 3.2. The individual rules and metric behaviour are discussed in the Supplementary Material that can be found at https://www.nature.com/articles/s41587-019-0071-9#Sec34.

3.5.3 Score aggregation

To rank the methods, we need to aggregate on two levels: across **datasets** and across specific/application metrics to calculate an **overall metric**.

Aggregating over datasets

When combining different datasets, it is important that the biases in the datasets does not influence the overall score. In our study, we define three such biases, although there are potentially many more:

- **Difficulty of the datasets** Some datasets are more difficult than others. This can have various reasons, such as the complexity of the topology, the amount of biological and technical noise, or the dimensions of the data. It is important that a small increase in performance on a more difficult dataset has an equal impact on the final score as a large increase in performance on easier datasets.

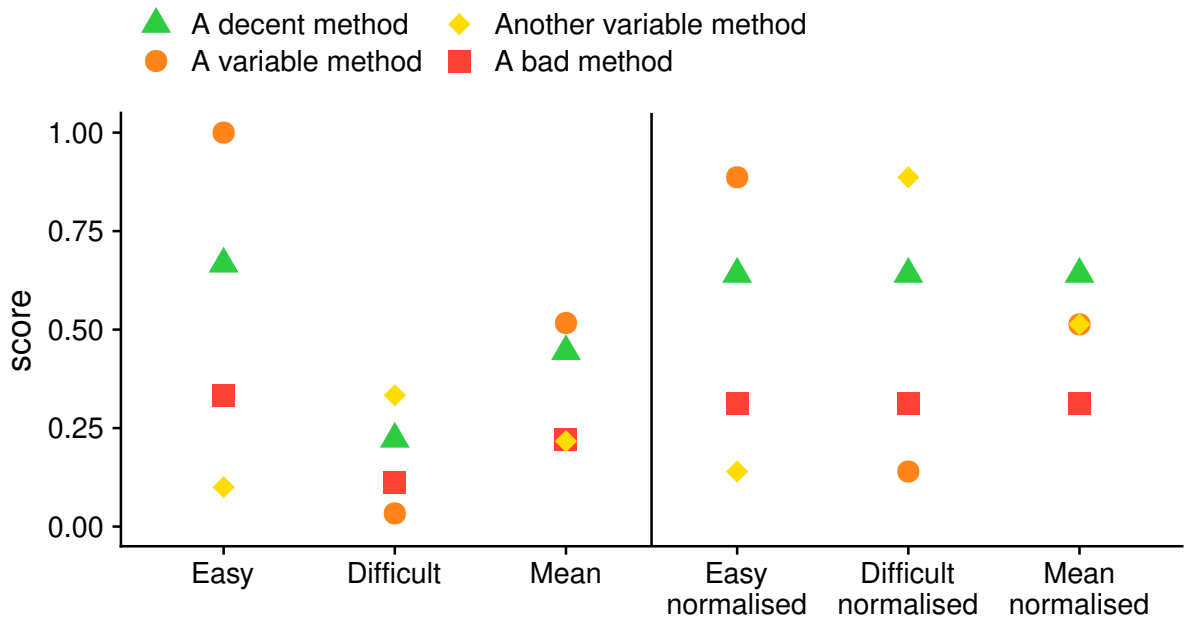


Figure 3.20: An illustration of how the difficulty of a dataset can influence the overall ranking. A decent method, which consistently ranks high on an easy and difficult dataset, does not get a high score when averaging. On the other hand, a method which ranks high on the easy dataset, but very low on the difficult dataset does get a high score on average. After normalising the scores (right), this problem disappears.

- **Dataset sources** It is much easier to generate synthetic datasets than real datasets, and this bias is reflected in our set of datasets. However, given their higher biological relevance, real datasets should be given at least equal importance than synthetic datasets.

- **Trajectory types** There are many more linear and disconnected real datasets, and only a limited number of tree or graph datasets. This imbalance is there because historically most datasets have been linear datasets, and because it is easy to create disconnected datasets by combining different datasets. However, this imbalance in trajectory types does not necessarily reflect the general importance of that trajectory type.

We designed an aggregation scheme which tries to prevent these biases from influencing the ranking of the methods.

The difficulty of a dataset can easily have an impact on how much weight the dataset gets in an overall ranking. We illustrate this with a simple example in Figure 3.20. One method consistently performs well on both the easy and the difficult datasets. But because the differences are small in the difficult datasets, the mean would not give this method a high score. Meanwhile, a variable method which does not perform well on the difficult dataset gets the highest score, because it scored so high on the easier dataset.

To avoid this bias, we normalise the scores of each dataset by first scaling and centering to $\mu = 0$ and $\sigma = 1$, and then moving the score values back to $[0, 1]$ by applying the unit normal density distribution function. This results in scores which are comparable across different datasets (Figure 3.20). In contrast to other possible normalisation techniques, this will still retain some information on the relative difference between the scores, which would have been lost when using the ranks for normalisation. An example of this normalisation, which will also be used in the subsequent aggregation steps, can be seen in Figure 3.21.

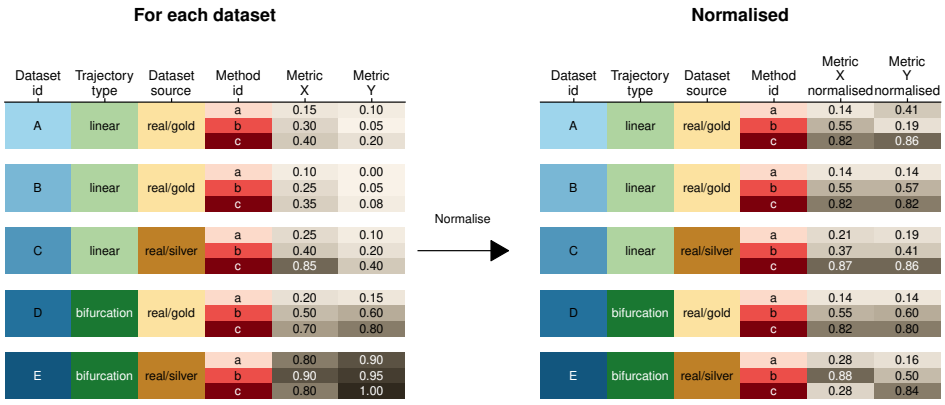


Figure 3.21: An example of the normalisation procedure. Shown are some results of a benchmarking procedure, where every row contains the scores of a particular method (red shading) on a particular dataset (blue shading), with a trajectory type (green shading) and dataset source (orange shading).

After normalisation, we aggregate step by step the scores from different datasets. We first aggregate the datasets with the same dataset source and trajectory type using an arithmetic mean of their scores (Figure 3.22a). Next, the scores are averaged over different dataset sources, using a arithmetic mean which was weighted based on how much the synthetic and silver scores correlated with the real gold scores (Figure 3.22b). Finally, the scores are aggregated over the different trajectory types again using a arithmetic mean (Figure 3.22c).

Overall metrics

Undoubtedly, a single optimal overall metric does not exist for trajectories, as different users may have different priorities:

- A user may be primarily interested in defining the correct topology, and only use the cellular ordering when the topology is correct
- A user may be less interested in how the cells are ordered within a branch, but primarily in which cells are in which branches
- A user may already know the topology, and may be primarily interested in finding good features related to a particular branching point
- ...

Each of these scenarios would require a combinations of *specific* and *application* metrics with different weights. To provide an "overall" ranking of the metrics, which is impartial for the scenarios described above, we therefore chose a metric which weighs every aspect of the trajectory equally:

- Its **ordering**, using the cor_{dist}
- Its **branch assignment**, using the $F1_{branches}$
- Its **topology**, using the *HIM*
- The accuracy of **differentially expressed features**, using the $wcor_{features}$

Next, we considered three different ways of averaging different scores: the arithmetic mean, geometric mean and harmonic mean. Each of these types of mean have different use cases. The harmonic mean is most appropriate when the scores would all have a common denominator (as is the

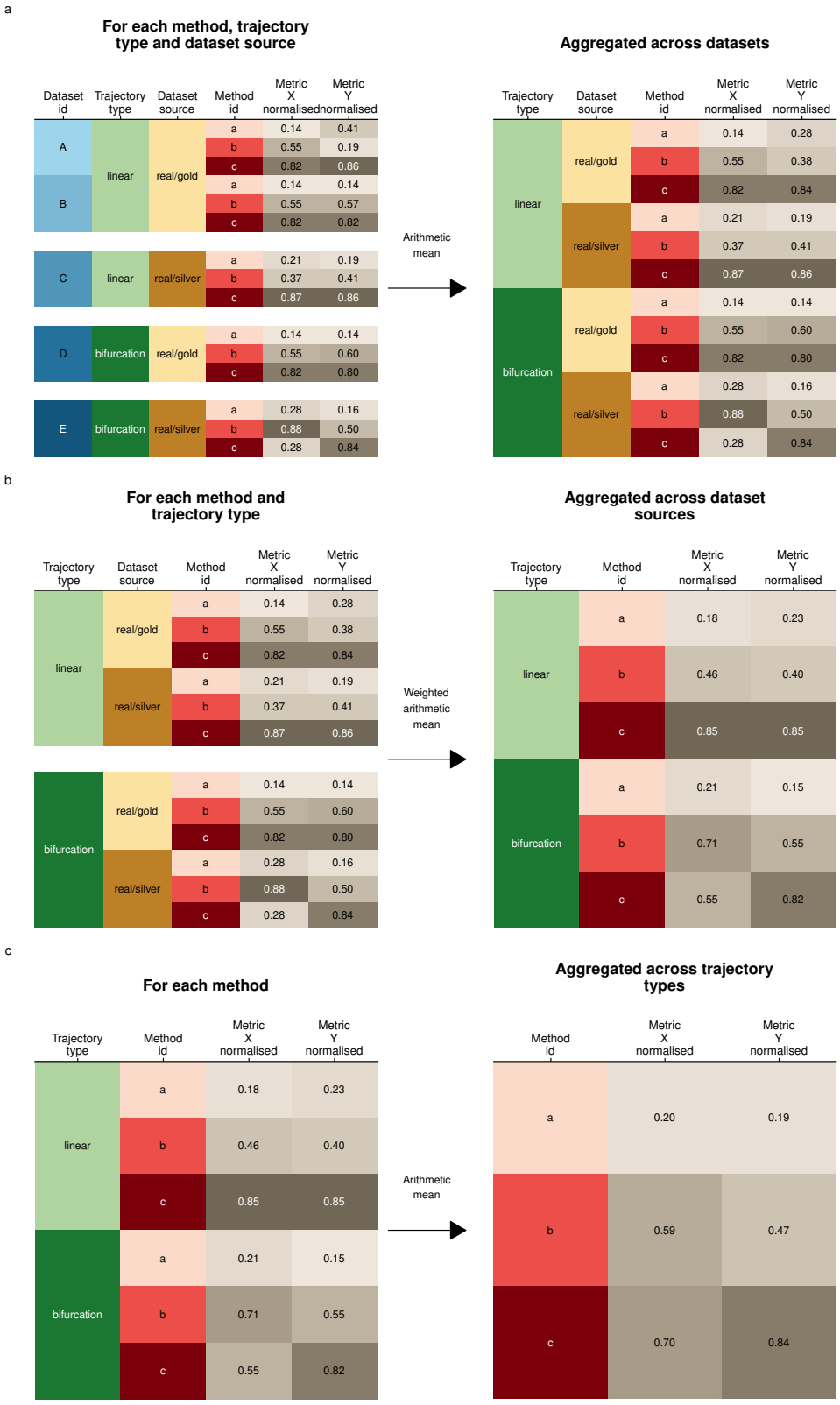


Figure 3.22: An example of the aggregation procedure. In consecutive steps we aggregated across **(a)** different datasets with the same source and trajectory type, **(b)** different dataset sources with the same trajectory type (weighted for the correlation of the dataset source with the real gold dataset source) and **(c)** all trajectory types.

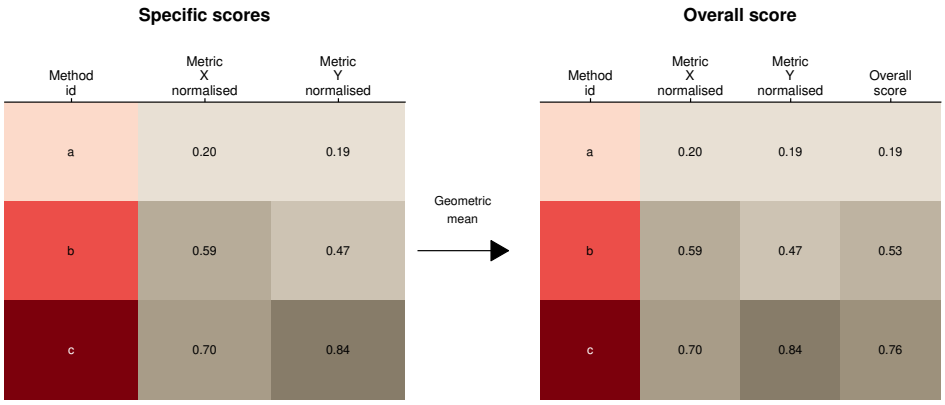


Figure 3.23: An example of the averaging procedure. For each method, we calculated the geometric mean between its normalised and aggregated scores

case for the *Recovery* and *Relevance* described earlier). The arithmetic mean would be most appropriate when all the metrics have the same range. For our use case, the geometric mean is the most appropriate, because it is low if one of the values is low. For example, this means that if a method is not good at inferring the correct topology, it will get a low overall score, even if it performs better at all other scores. This ensures that a high score will only be reached if a prediction has a good ordering, branch assignment, topology, and set of differentially expressed features.

The final overall score (Figure 3.23) for a method was thus defined as:

$$Overall = mean_{geometric} = \sqrt[4]{cor_{dist} \times F1_{branches} \times HIM \times wcor_{features}}$$

We do however want to stress that different use cases will require a different overall score to order the methods. Such a context-dependent ranking of all methods is provided through the dynguidelines app (http://guidelines.dynverse.org).

CHAPTER 4

dyno: A toolkit for inferring and interpreting trajectories

Abstract: Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Adapted from:
Cannoodt, R., Saelens, W., and Saeys, Y. dyno: A toolkit for inferring and interpreting trajectories. *Journal* vol, issue (2019), page–page. doi.

4

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

CHAPTER 5

SCORPIUS: Fast, accurate, and robust single-cell pseudotime

Abstract: Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Adapted from:
Cannoodt, R., ..., De Preter, K., and Saeys, Y. SCORPIUS improves trajectory inference and identifies novel modules in dendritic cell development. *Journal* vol, issue (2019), page–page. doi:10.1101/079509v2.

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

CHAPTER 6

bred: Inferring single cell regulatory networks

Abstract: Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Adapted from:
Cannoodt, R., Saelens, W., and Saeys, Y. Inferring Single Cell Regulatory Networks. *Journal* vol, issue (2019), page–page. doi.

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

CHAPTER 7

incgraph: Optimising regulatory networks

Abstract: Graphlets are small network patterns that can be counted in order to characterise the structure of a network (topology). As part of a topology optimisation process, one could use graphlet counts to iteratively modify a network and keep track of the graphlet counts, in order to achieve certain topological properties. Up until now, however, graphlets were not suited as a metric for performing topology optimisation; when millions of minor changes are made to the network structure it becomes computationally intractable to recalculate all the graphlet counts for each of the edge modifications.
IncGraph is a method for calculating the differences in graphlet counts with respect to the network in its previous state, which is much more efficient than calculating the graphlet occurrences from scratch at every edge modification made. In comparison to static counting approaches, our findings show IncGraph reduces the execution time by several orders of magnitude. The usefulness of this approach was demonstrated by developing a graphlet-based metric to optimise gene regulatory networks. IncGraph is able to quickly quantify the topological impact of small changes to a network, which opens novel research opportunities to study changes in topologies in evolving or online networks, or develop graphlet-based criteria for topology optimisation.
IncGraph is freely available as an open-source R package on CRAN (incgraph). The development version is also available on GitHub (rcannood/incgraph).

Adapted from:
Cannoodt, R., Ruyssinck, J., Ramon, J., De Preter, K., and Saeys, Y. IncGraph: Incremental graphlet counting for topology optimisation. *PLOS ONE* 13, 4 (2018), e0195997. doi:10.1371/journal.pone.0195997.

7

7.1 Introduction

Even the simplest of living organisms already consist of complex biochemical networks which must be able to respond to a variety of stressful conditions in order to survive. An organism can be characterised using numerous interaction networks, including gene regulation, metabolic, signalling, and protein-protein interaction. The advent of high-throughput profiling methods (e.g. microarrays and RNA sequencing) have allowed to observe the molecular contents of a cell, which in turn have enabled computational network inference methods to reverse engineer the biochemical interaction networks [109]. Improving the accuracy of inferred networks has been a long-standing challenge, but the development of ever more sophisticated algorithms and community-wide benchmarking studies have resulted in significant process [29, 110, 30, 111].

Several recent developments involve incorporating topological priors, either to guide the inference process [112] or post-process the network [113]. A common prior is that biological networks are highly modular [114], as they consist of multiple collections of functionally or physically linked molecules [115, 116]. At the lowest level, each module is made up out of biochemical interactions arranged in small topological patterns, which act as fundamental building blocks [117].

Graphlets [118] are one of the tools which could be used to add a topological prior to a biological network, Graphlets are small connected subnetworks which can be counted to identify which low-level topological patterns are present in a network. By comparing how topologically similar a predicted network is to what would be expected of a true biological network, a predicted network can be optimised in order to obtain a better topology.

By counting the number of occurrences of each of the different graphlets (Fig 7.1A) touching a specific node, one can characterise the topology surrounding it. The graphlet counts of a network can be represented as a matrix with one row for each of the nodes and one column for each of the graphets (Fig 7.1B). An orbit represents a class of isomorphic (i.e. resulting in the same structure) positions of nodes within a graphlet (Fig 7.1A, coloured in red). Both graphlets and orbits have been used extensively for predicting the properties of nodes such as protein functionality [119, 120, 121] and gene oncogenicity [122], by performing network alignment [123, 124] or using them as a similarity measure in machine learning tasks [125, 126].

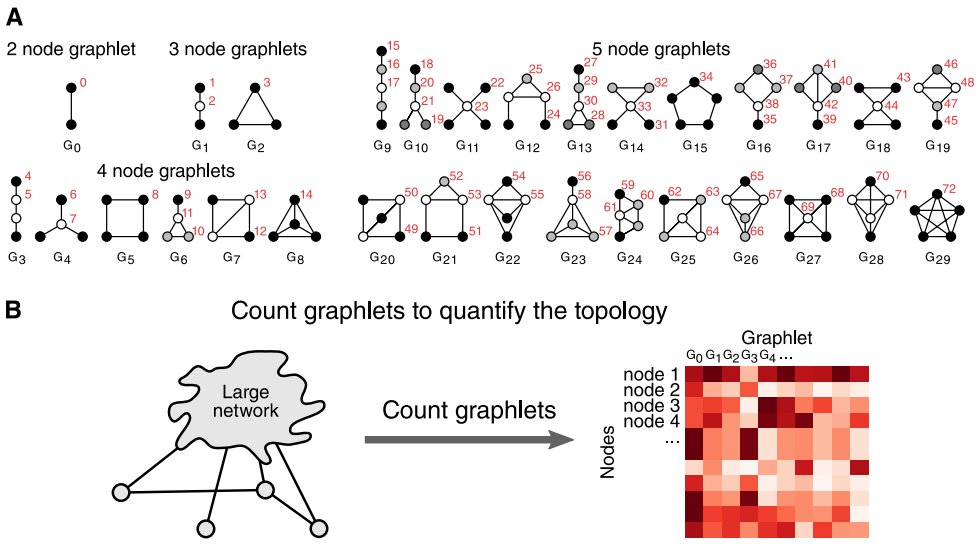


Figure 7.1: Graphlet counting in a network characterises its local topologies. (A) In total, there are 30 different graphlets containing 2 to 5 nodes, ranging from G_0 to G_{29}. Orbits are an extension of graphlets which also take into account the position of a node within a graphlet. The 73 different orbits are coloured in red. (B) By counting the occurrences of these graphlets in the network, the local topology surrounding a node can be quantified.

In this work, we focus on optimising gene regulatory networks by incorporating a topological prior as part of a topology optimisation process. We seek to optimise a predicted network by iteratively modifying the network and accepting modifications that lead to topological properties that resemble better those of true biological networks.

However, using graphlets to perform topology optimisation was hitherto not possible. Calculating the graphlet counts using the most state-of-the-art graphlet counting of a moderately sized gene regulatory network already has an execution time of about five seconds (*E. coli*, ~ 3000 genes, ~ 10000 interactions, up to graphlets up to 5 nodes). While this computational time poses no issue for regular static networks, recalculating all graphlet counts for every network modification made as part of a topology optimisation is computationally intractable. For example, performing 100'000 iterations of topology optimisation on a similarly sized network and calculating the topological impact of 10 possible edge modification at each iteration, already results in a computational time of about 12 days. Graphlet-based topology optimisation thus quickly becomes infeasible for larger networks.

When adding or removing an edge to a large network, the number of altered graphlets is much smaller than the total number of graphlets in the network. It could therefore be much more efficient to iterate over and count all the graphlets that have been added or removed as a result of the edge modification, than it is to recalculate the graphlet counts from scratch.

Eppstein et al. introduced data structures and algorithms for updating the counts of size-three[127] and size-four[128] subgraphs in a dynamic setting. The data structures were determined such that the amortised time is $O(h)$ and $O(h^2)$, respectively, where h is the h-index of the network[129].

We developed IncGraph, an alternative algorithm and implementation for performing incremental counting of graphlets up to size five. We show empirically that IncGraph is several orders of magnitude faster at calculating the differences in graphlet counts in comparison to non-incremental counting approaches. In addition, we demonstrate the practical applicability by developing a graphlet-based optimisation criterion for biological networks.

7.2 Materials and methods

Assume a network G of which the graphlet counts C_G are known. C_G is an n-by-m matrix, with n the number of vertices in the network, $m = 73$ is the number of different orbits, and where $C_G[i, j]$ is the number of times node i is part of a graphlet at orbit O_j. Further assume that one edge has either been added or removed from G, resulting in G', at which point the counts $C_{G'}$ need to be observed. If multiple edges have been modified, the method described below can be repeated for each edge individually.

7.2.1 Incremental graphlet counting

As stated earlier, recalculating the graphlet counts for each modification made to the network quickly becomes computationally intractable for larger network sizes. However, as the differences in topology between G and G' are small, it is instead possible to calculate the differences in graphlet counts $\Delta_{G,G'}$ instead. This is much more efficient to calculate, as only the neighbourhood of the modified edges needs to be explored. $C_{G'}$ can thus be calculated as $C_{G'} = C_G + \Delta_{G,G'}$.

The differences in graphlet counts $\Delta_{G,G'}$ are calculated by iterating recursively over the neighbours surrounding each of the modified edges (See S1 Pseudocode). Several strategies are used in order to calculate $\Delta_{G,G'}$ as efficiently as possible (Fig 7.2). (A) The delta matrix is calculated separately for each modified edge. Since the edge already contains two out of five nodes and any modified graphlet is a connected subgraph, the neighbourhood of this edge only needs to be explored up to depth 3 in

order to iterate over all modified graphlets. (B) We propose a lookup table to look up the graphlet index of each node of a given subgraph. By weighting each possible edge in a 5-node graphlet, the sum of the edges of a subgraph can be used to easily look up the corresponding graphlet index. (C) During the recursive iteration of the neighbourhood, the same combination of nodes is never visited twice. (D) As the network can be relatively large, the adjacency matrix is binary compressed in order to save memory. One integer requires 4 bytes and contains the adjacency boolean values of 32 edges, whereas otherwise 32 booleans would require 32 bytes. For example, 1GB of memory is large enough to store a compressed adjacency matrix of 92681 nodes. If the network contains too many nodes to fit a compressed adjacency matrix into the memory, a list of sets containing each node's neighbours is used instead.

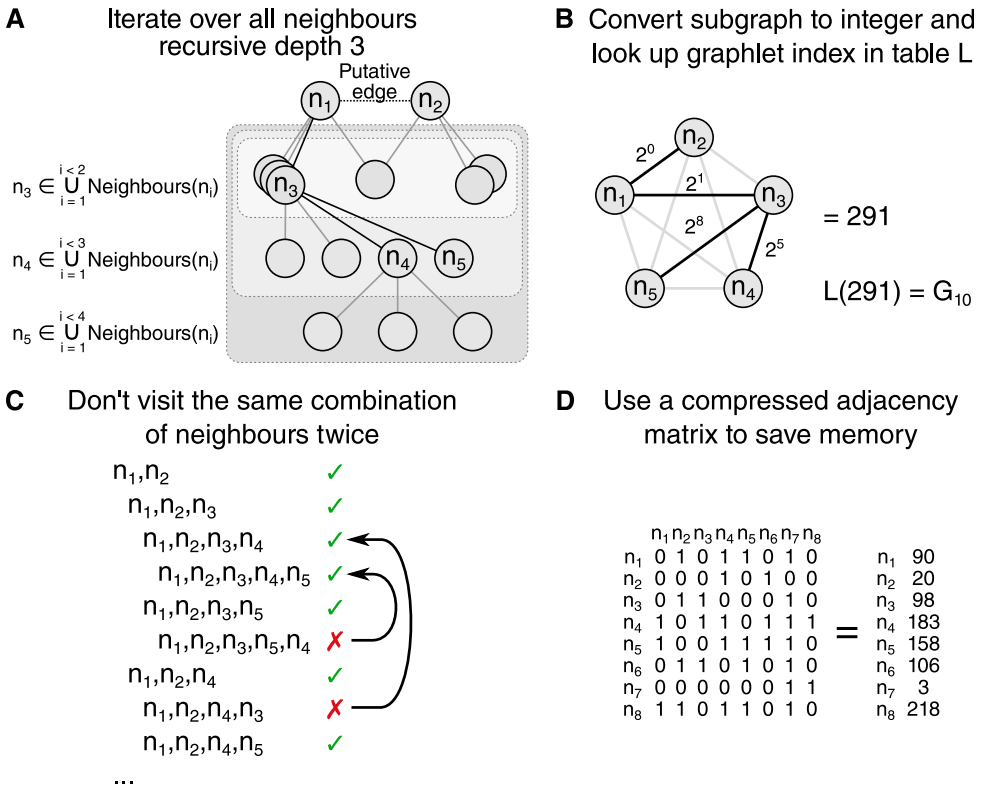


Figure 7.2: Several strategies are employed in order to reduce time and memory consumption. (A) Only the depth 3 neighbourhood of each modified edge needs to be explored in order to have visited all modified five-node graphlets. (B) A lookup table can be used to easily look up the graphlet index of a subgraph, by weighing each edge in a 5-node subgraph by a power of 2. (C) The same combination of five nodes is never repeated, as to avoid counting the same graphlet multiple times. (D) The adjacency matrix of the network is compressed in order to reduce memory usage.

IncGraph supports counting graphlets and orbits of subgraphs up to five nodes in undirected networks. By modifying the lookup table, the method can be easily extended to directed graphlets or larger-node graphlets, or limited to only a selection of graphlets. This allows for variations of the typical graphlets to be studied in an incremental setting.

7.2.2 Timing experiments

We compared the execution time needed to calculate the graphlet counts in iteratively modified networks between our method and a state-of-the-art non-incremental algorithm, Orca [130]. Orca is a heavily optimised algorithm for counting 5-node graphlets in static networks, and outperforms all other static graphlet counting algorithms by an order of magnitude [130].

The timing experiments were performed by generating networks from 3 different network models, making modifications to those networks while still adhering to the network model, and measuring the execution times taken for both approaches to calculate the new graphlet counts (Fig 7.3).

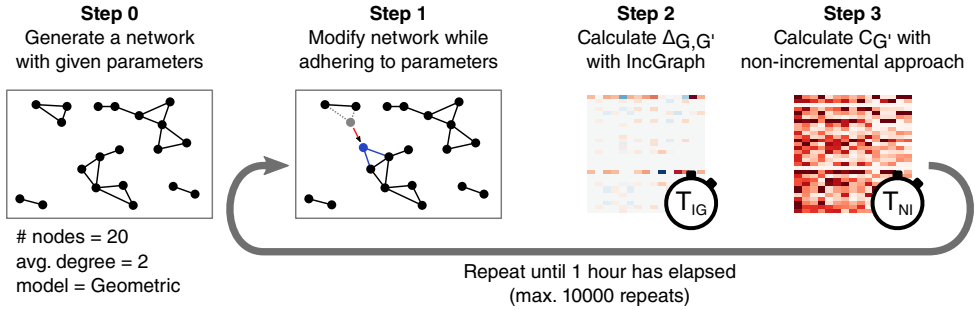


Figure 7.3: Static network model generators were modified to generate dynamic networks. Three network models were used: Barabási–Albert, Erdős–Rényi, and Geometric. Step 0: a network is generated according to the network model and the given parameters. Step 1: the network is modified such that the result is as likely to have been generated by the network model. Step 2: The time for IncGraph to calculate the differences in graphlet counts is measured (T_{IG}). Step 3: The time for the non-incremental approach to calculate the graphlet counts of the new network is measured (T_{NI}). Steps 1 to 3 are repeated until all modifications generated at step 0 are processed, or until an execution time threshold has been reached.

The network models were based on three static network models: Barabási-Albert [131], Erdős–Rényi [132], and Geometric [133]. Pseudo code for these random evolving network models can be found in S2 Pseudocode, S3 Pseudocode, and S4 Pseudocode respectively. Each model generates an initial network according to the static network model it is based on, and a list of network modifications (removing an edge from or adding an edge to the network). These network modifications are made such that at any given time point in the evolving network, it is likely that the network at its current state could have been generated by the original static network model.

Networks were generated with varying network models, between 1000 and 16000 nodes, node degrees between 2 and 64, and 10000 time points. We measured the time needed to calculate the delta matrix at random time points until 1 hour has passed. All timings experiments were carried out on Intel(R) Xeon(R) CPU E5-2665 @ 2.40GHz processors, with one thread per processor. The generation of networks with higher node counts or degrees was constrained by the execution time of the network generators, not by IncGraph. All data and scripts are made available at github.com/rcannood/incgraph-scripts.

7.2.3 Gene regulatory network optimisation experiments

We demonstrate the usefulness of IncGraph by using a simple graphlet-based metric to optimise gene regulatory networks. One of the striking differences between real and predicted gene regulatory networks is that the predicted networks contain highly connected subnetworks, which contain high amounts of false positives. We determine a penalty score such that predicted networks containing graphlets with many redundant edges will be penalised in comparison to very sparse networks.

The *redundancy penalty* (Fig 7.4A) of a network is defined as the sum of occurrences of each graphlet multiplied by the redundancy associated with each individual graphlet. The redundancy of a graphlet is the number of edges that can be removed without disconnecting the nodes from one another. By using the redundancy penalty score, we aim to improve the gene regulatory network (Fig 7.4B).

The topology optimisation procedure uses an empty network as initialisation and grows the network by selecting interactions iteratively. Each iteration, the top $k = 100$ highest ranked interactions

that are not currently part of the network are evaluated, and the highest ranked interaction passing the redundancy criterion is selected (Fig 7.4C). This procedure is repeated until a predefined amount of time has passed. As the aim of this experiment is not to obtain the highest performing topology optimisation method, parameter optimisation of k has not been performed and is considered to be outside the scope of this work.

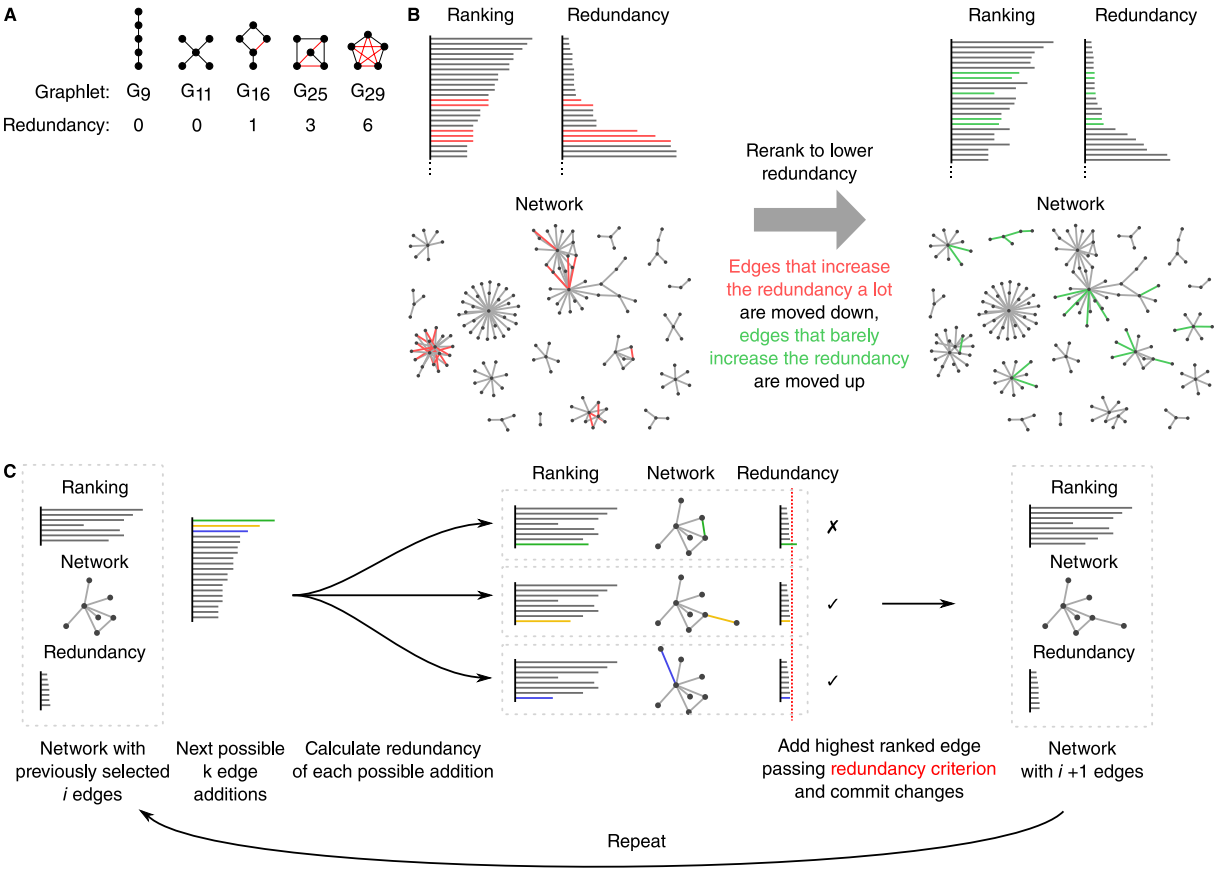


Figure 7.4: Predicted gene regulatory networks of model organisms are optimised to reduce the false positive rate. A) The number of redundant edges in each graphlet are counted. B) The network is optimised in order to obtain a lower redundancy over time. Two networks are shown, one before and one after the optimisation procedure. Edges coloured in red have been removed from the network after optimisation, green edges have been added. C) Starting from an empty network, the interactions are modified by iteratively evaluating the increase in redundancy of the next k interactions, and adding the first edge for which its redundancy is less than the 90^{th} percentile redundancy.

We optimised gene regulatory networks of *E. coli* and *S. cerevisiae*. The predicted networks were generated using the network inference method GENIE3 [134] with default parameters. Gene expression data was obtained from COLOMBOS [135] and GEO [136], respectively. The predicted networks and the optimised versions thereof were compared against respective lists of known gene regulatory interactions [137, 138].

7.3 Results and discussion

The contributions of this work are twofold. Firstly, we propose a new method for incrementally calculating the differences in graphlet counts in changing graphs, and show that it is orders of magnitude faster than non-incremental approaches. Secondly, we demonstrate its applicability by optimising a predicted gene regulatory network in order to reduce the false positive rate therein.

7.3.1 Execution time is reduced by orders of magnitude

Timing experiments show that IncGraph is significantly faster in calculating the delta matrix in comparison to calculating the graphlet counts from scratch at each iteration (Fig 7.5). The observed speedup ratios between IncGraph and the non-incremental approach Orca ranges from about 50× to 10000×. The speedup ratio increases as the network size increases. For larger networks, IncGraph can thus calculate the delta matrices of 10000 edge modifications while the non-incremental approach calculates one graphlet count matrix.

Surprisingly, IncGraph obtains higher execution times for networks with 5657 nodes than for networks with 8000 nodes. One possible explanation is that the size of the data structures containing those networks are particularly favourable in avoiding cache misses. Confirmation of this explanation, however, would require further investigation.

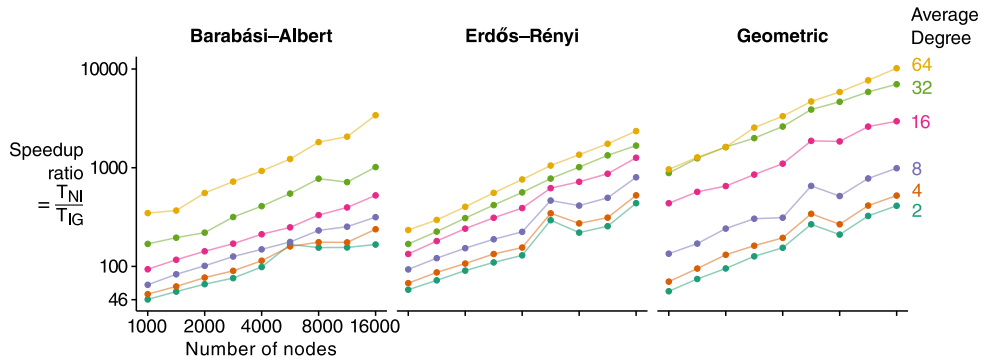


Figure 7.5: IncGraph is significantly faster than non-incremental approaches. For small networks, the execution time of IncGraph T_{IG} is already 50 times less than that of non-incremental approaches T_{NI}. This ratio increases even further for networks with higher numbers of nodes or higher average degrees.

Comparing the execution time of IncGraph to the h-index of the networks indicates that the amortised time of IncGraph could be $O(h^3)$ (S1 Fig). This is in line with the amortised times $O(h)$ and $O(h^2)$ of the algorithm described by Eppstein et al.[128] for counting three-size and four-size subgraphs respectively.

7.3.2 IncGraph allows for better regulatory network optimisation

We implemented a graphlet-based optimisation algorithm for improving the false positive rate of the predicted gene regulatory networks of *E. coli* and *S. cerevisiae*. After reranking the regulatory interactions, the F1 score of the first 1000 interactions had increased by 7.6% and 2.2% respectively (Fig 7.6A). The obtained speedup of about 15-30× (Fig 7.6B) is in line with the experiments on *in silico* networks. Namely, for the *E. coli* network at 9618 interactions and 889 nodes (average degree = 10.8), a speedup of about 30× was obtained. Similarly, for the *S. cerevisiae* network at 8013 interactions and 1254 nodes (average degree = 6.4), a speedup of about 15× was obtained. These speedups are in the same order of magnitude of similarly sized networks (1000 nodes and 8000 interactions) generated with a Barabási-Albert model, with a speedup of 65×. This is to be expected, as such networks share the same scale-free property that gene regulatory networks have.

7.4 Conclusion

Many improvements over the past few years have resulted in efficient graphlet counting algorithms, even for large static networks. However, needing to perform the simplest of tasks tens of thousands

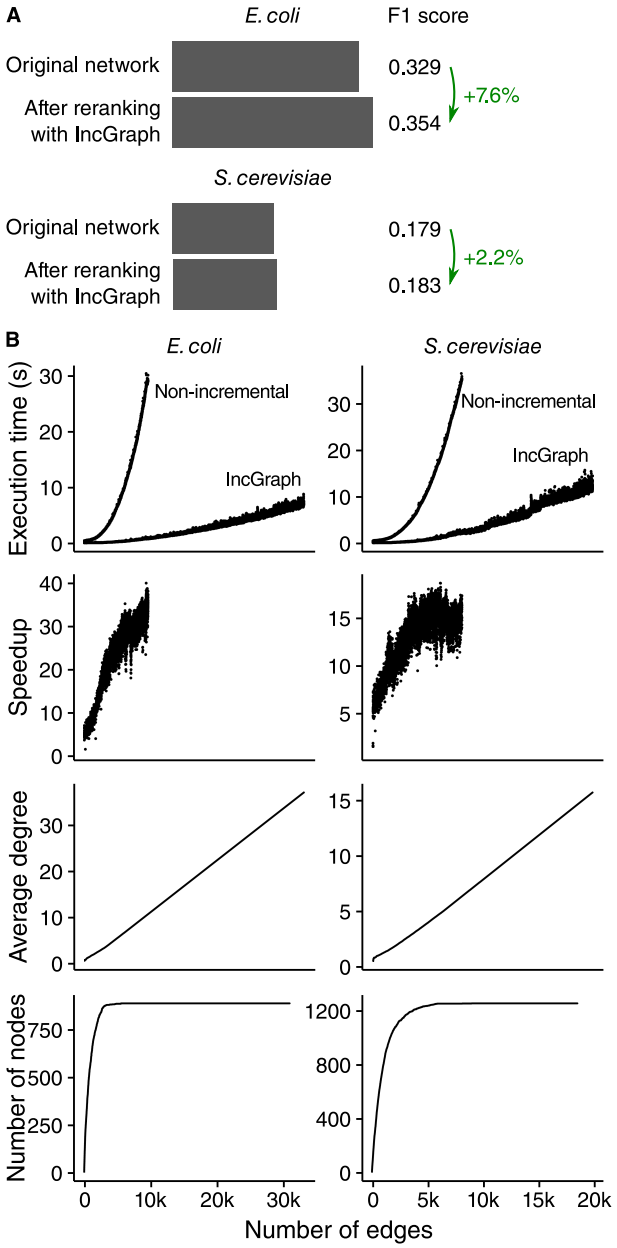


Figure 7.6: A simple graphlet-based scoring method improves predicted regulatory networks. (A) The F1 score was calculated by calculating the harmonic mean of the AUROC and AUPR scores of the first 1000 interactions. (B) IncGraph is significantly faster than the non-incremental approach. Note that for each interaction added to the network, the graphlet counts of 100 putative interactions were evaluated.

of times quickly becomes computationally intractable. As such, recalculating the graphlet counts of a network after each of the many network modification is infeasible.

This study introduces a method for calculating the differences in graphlet (and orbit) counts, which we call incremental graphlet counting or IncGraph for short. We show that IncGraph is at least 10-100 times faster than non-incremental methods for networks of moderate size, and that the speedup ratio increases even further for larger networks. To demonstrate the applicability of IncGraph, we optimised a predicted gene regulatory network by enumerating over the ranked edges and observing the graphlet counts of several candidate edges before deciding which edge to add to the network.

IncGraph enables graphlet-based metrics to characterize online networks, e.g. in order to track certain network patterns over time, as a similarity measure in a machine learning task, or as a criterion in a topology optimisation.

7.5 Supporting information

S1 Pseudocode. IncGraph calculates $\Delta_{G,G'}$ using a strict branch-and-bound strategy.

S2 Pseudocode. Pseudo code for generating an evolving Barabási-Albert (BA) network. It first generates a BA network, and then generates o operations such that at any time point, the network is or very closely resembles a BA network.

S3 Pseudocode. Pseudo code for generating an evolving Erdős–Rényi (ER) network. It first generates an ER network, and then generates o operations such that at any time point, the network is or very closely resembles an ER network.

S4 Pseudocode. Pseudo code for generating an evolving geometric network. It first generates a geometric network, and then generates o operations such that at any time point, the network is or very closely resembles a geometric network.

S1 Fig. Empirical measurements show a strong relation between the execution time of IncGraph and the h-index cubed of the network it was applied to. This is in line with the findings by Eppstein et al., where counting 3-size subgraphs has an amortised time of $O(h)$ and counting 4-size subgraphs has an amortised time of $O(h^2)$.

CHAPTER 8

General discussion

8

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

8.1 Overview and conclusions of the presented work

8.2 Future research directions

Samenvatting

Summary

List of Publications

Bibliography

[1] Ingo Brigandt and Alan Love. "Reductionism in Biology". In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N Zalta. Spring 201. Published: $\backslash$url{https://plato.stanford.edu/archives/spr2 biology/}. Metaphysics Research Lab, Stanford University, 2017.

[2] Aviv Regev et al. "The Human Cell Atlas White Paper". In: (Oct. 2018).

[3] Human Cell Atlas consortium. *Human Cell Atlas Data Portal*. 2018.

[4] Chung Chau Hon et al. "The Human Cell Atlas: Technical Approaches and Challenges". In: *Briefings in Functional Genomics* 17.4 (July 2018), pp. 283–294. ISSN: 20412657. DOI: 10.1093/bfgp/elx029.

[5] James D Watson, Francis HC Crick, et al. "Molecular Structure of Nucleic Acids". In: *Nature* 171.4356 (1953), pp. 737–738.

[6] Bruce Alberts et al. "The RNA World and the Origins of Life". en. In: *Molecular Biology of the Cell. 4th edition* (2002).

[7] David P. Horning. "RNA World". In: *Encyclopedia of Astrobiology*. Ed. by Muriel Gargaud et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 1466–1478. ISBN: 978-3-642-11274-4. DOI: 10.1007/978-3-642-11274-4_1740.

[8] Olga Kelemen et al. "Function of Alternative Splicing". In: *Gene* 514.1 (Feb. 2013), pp. 1–30. ISSN: 0378-1119. DOI: 10.1016/j.gene.2012.07.083.

[9] Albert H Coons, Hugh J Creech, and R Norman Jones. "Immunological Properties of an Antibody Containing a Fluorescent Group." In: *Proceedings of the Society for Experimental Biology and Medicine* 47.2 (1941), pp. 200–202.

[10] M. J. Fulwyler. "Electronic Separation of Biological Cells by Volume". In: *Science* 150.3698 (1965), pp. 910–911. ISSN: 0036-8075. DOI: 10.1126/science.150.3698.910.

[11] Satya P. Yadav. "The Wholeness in Suffix -Omics, -Omes, and the Word Om". In: *Journal of Biomolecular Techniques : JBT* 18.5 (Dec. 2007), p. 277. ISSN: 1524-0215.

[12] Fuchou Tang et al. "mRNA-Seq Whole-Transcriptome Analysis of a Single Cell". en. In: *Nature Methods* 6.5 (May 2009), pp. 377–382. ISSN: 1548-7105. DOI: 10.1038/nmeth.1315.

[13] Jan Philipp Junker and Alexander van Oudenaarden. "Every Cell Is Special: Genome-Wide Studies Add a New Dimension to Single-Cell Biology". In: *Cell* 157.1 (Mar. 2014), pp. 8–11. ISSN: 0092-8674. DOI: 10.1016/j.cell.2014.02.010.

[14] Arnav Moudgil. *Multimodal scRNA-Seq*. Feb. 2019. DOI: 10.5281/zenodo.2628012.

[15] Oliver Stegle, Sarah A. Teichmann, and John C. Marioni. "Computational and Analytical Challenges in Single-Cell Transcriptomics". en. In: *Nature Reviews Genetics* 16.3 (Mar. 2015), pp. 133–145. ISSN: 1471-0064. DOI: 10.1038/nrg3833.

[16] Guo-Cheng Yuan et al. "Challenges and Emerging Directions in Single-Cell Analysis". In: *Genome Biology* 18.1 (May 2017), p. 84. ISSN: 1474-760X. DOI: 10.1186/s13059-017-1218-y.

[17] Geng Chen, Baitang Ning, and Tieliu Shi. "Single-Cell RNA-Seq Technologies and Related Computational Data Analysis". English. In: *Frontiers in Genetics* 10 (2019). ISSN: 1664-8021. DOI: 10.3389/fgene.2019.00317.

[18] Allon Wagner, Aviv Regev, and Nir Yosef. "Revealing the Vectors of Cellular Identity with Single-Cell Genomics". en. In: *Nature Biotechnology* 34.11 (Nov. 2016), pp. 1145–1160. ISSN: 1546-1696. DOI: 10.1038/nbt.3711.

[19] Luke Zappia, Belinda Phipson, and Alicia Oshlack. "Exploring the Single-Cell RNA-Seq Analysis Landscape with the scRNA-Tools Database". en. In: *PLOS Computational Biology* 14.6 (June 2018), e1006245. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1006245.

[20] Daniel Engel, Lars Hüttenberger, and Bernd Hamann. "A Survey of Dimension Reduction Methods for High-Dimensional Data Analysis and Visualization". In: *Visualization of Large and Unstructured Data Sets: Applications in Geospatial Planning, Modeling and Engineering - Proceedings of IRTG 1131 Workshop 2011*. Ed. by Christoph Garth, Ariane Middel, and Hans Hagen. Vol. 27. OpenAccess Series in Informatics (OASIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2012, pp. 135–149. ISBN: 978-3-939897-46-0. DOI: 10.4230/OASIcs.VLUDS.2011.135.

[21] Amos Tanay and Aviv Regev. "Scaling Single-Cell Genomics from Phenomenology to Mechanism". In: *Nature* 541.7637 (Jan. 2017), nature21350. ISSN: 1476-4687. DOI: 10.1038/nature21350.

[22] Martin Etzrodt, Max Endele, and Timm Schroeder. "Quantitative Single-Cell Approaches to Stem Cell Research". In: *Cell Stem Cell* 15.5 (2014), pp. 546–558.

[23] 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.

[24] Aviv Regev et al. "The Human Cell Atlas". In: *eLife* 6 (Dec. 2017). ISSN: 2050084X. DOI: 10.7554/eLife.27041.

[25] Xiaoping Han et al. "Mapping the {{Mouse Cell Atlas}} by {{Microwell}}-{{Seq}}". In: *Cell* 172.5 (Feb. 2018), 1091–1107.e17. ISSN: 1097-4172. DOI: 10.1016/j.cell.2018.02.001.

[26] Nicholas Schaum et al. "Single-Cell Transcriptomics of 20 Mouse Organs Creates a {{Tabula Muris}}". In: *Nature* 562.7727 (Oct. 2018), pp. 367–372. ISSN: 1476-4687. DOI: 10.1038/s41586-018-0590-4.

[27] 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.

[28] Philipp Angerer et al. "Single Cells Make Big Data: {{New}} Challenges and Opportunities in Transcriptomics". In: *Current Opinion in Systems Biology*. Big Data Acquisition and Analysis $\backslash$textbullet{} Pharmacology and Drug Discovery 4 (Aug. 2017), pp. 85–91. ISSN: 2452-3100. DOI: 10.1016/j.coisb.2017.07.004.

[29] Daniel Marbach et al. "Revealing Strengths and Weaknesses of Methods for Gene Network Inference". In: *Proceedings of the {N}ational {A}cademy of {S}ciences* 107.14 (Apr. 2010), pp. 6286–6291. ISSN: 1091-6490. DOI: 10.1073/pnas.0913357107.

[30] 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.

[31] Olivia Padovan-Merhar and Arjun Raj. "Using Variability in Gene Expression as a Tool for Studying Gene Regulation". eng. In: *Wiley Interdisciplinary Reviews. Systems Biology and Medicine* 5.6 (Nov. 2013), pp. 751–759. ISSN: 1939-005X. DOI: 10.1002/wsbm.1243.

[32] Atefeh Lafzi et al. "Tutorial: Guidelines for the Experimental Design of Single-Cell RNA Sequencing Studies". In: *Nature Protocols* 13.12 (Dec. 2018), pp. 2742–2757. ISSN: 1750-2799. DOI: 10.1038/s41596-018-0073-y.

[33] Malte D Luecken and Fabian J Theis. "Current Best Practices in Single-Cell RNA-Seq Analysis: A Tutorial". In: *Molecular Systems Biology* 15.6 (June 2019), e8746. ISSN: 1744-4292. DOI: 10.15252/msb.20188746.

[34] Vladimir Kiselev et al. *Analysis of Single Cell RNA-Seq Data*. English. Cambridge, UK, May 2019.

[35] Liesbet Martens and Niels Vandamme. *Analysis of Single Cell RNA-Seq Data from 10x Genomics*. English. Ghent, Aug. 2019.

[36] Martin Hemberg. *Coffee Break during "Analysis of Single Cell RNA-Seq Data 23-24 May 2019" Workshop*. May 2019.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] Cole Trapnell. "Defining Cell Types and States with Single-Cell Genomics". In: *Genome Research* 25.10 (2015), pp. 1491–1498. ISSN: 15495469. DOI: 10.1101/gr.190595.115.

[42] Kevin R Moon et al. "Manifold Learning-Based Methods for Analyzing Single-Cell {{RNA}}-Sequencing Data". In: *Current Opinion in Systems Biology*. $\backslash$textbullet{} Future of Systems Biology$\backslash$textbullet{} Genomics and Epigenomics 7 (Feb. 2018), pp. 36–46. ISSN: 2452-3100. DOI: 10.1016/j.coisb.2017.12.008.

[43] Zehua Liu et al. "Reconstructing Cell Cycle Pseudo Time-Series via Single-Cell Transcriptome Data". In: *Nature Communications* 8.1 (June 2017), p. 22. ISSN: 2041-1723. DOI: 10.1038/s41467-017-00039-z.

[44] F Alexander Wolf et al. "Graph Abstraction Reconciles Clustering with Trajectory Inference through a Topology Preserving Map of Single Cells". In: *bioRxiv* (Oct. 2017), p. 208819. DOI: 10.1101/208819.

[45] Andreas Schlitzer et al. "Identification of {{cDC1}}- and {{cDC2}}-Committed {{DC}} Progenitors Reveals Early Lineage Priming at the Common {{DC}} Progenitor Stage in the Bone Marrow". In: *Nature Immunology* 16.7 (July 2015), pp. 718–728. ISSN: 1529-2916. DOI: 10.1038/ni.3200.

[46] Lars Velten et al. "Human Haematopoietic Stem Cell Lineage Commitment Is a Continuous Process". In: *Nature Cell Biology* 19.4 (Apr. 2017), pp. 271–281. ISSN: 1476-4679. DOI: 10.1038/ncb3493.

[47] Peter See et al. "Mapping the Human {{DC}} Lineage through the Integration of High-Dimensional Techniques". In: *Science* 356.6342 (June 2017), eaag3009. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.aag3009.

[48] Vincent J Henry et al. "{{OMICtools}}: An Informative Directory for Multi-Omic Data Analysis". In: *Database: The Journal of Biological Databases and Curation* 2014 (July 2014). ISSN: 1758-0463. DOI: 10.1093/database/bau069.

[49] Sean Davis et al. *Awesome Single Cell*. https://github.com/seandavi/awesome-single-cell. June 2018. DOI: 10.5281/zenodo.1294021.

[50] Luke Zappia, Belinda Phipson, and Alicia Oshlack. "Exploring the Single-Cell {{RNA}}-Seq Analysis Landscape with the {{scRNA}}-Tools Database". In: *bioRxiv* (Oct. 2017), p. 206573. DOI: 10.1101/206573.

[51] Sean C. Bendall et al. "Single-Cell Trajectory Detection Uncovers Progression and Regulatory Coordination in Human B Cell Development". In: *Cell* 157.3 (2014), pp. 714–725. ISSN: 00928674. DOI: 10.1016/j.cell.2014.04.005.

[52] Jaehoon Shin et al. "Single-Cell RNA-Seq with Waterfall Reveals Molecular Cascades Underlying Adult Neurogenesis". eng. In: *Cell Stem Cell* 17.3 (Sept. 2015), pp. 360–372. ISSN: 1875-9777. DOI: 10.1016/j.stem.2015.07.013.

[53] Kieran Campbell and Christopher Yau. "Bayesian {{Gaussian Process Latent Variable Models}} for Pseudotime Inference in Single-Cell {{RNA}}-Seq Data". In: *bioRxiv* (Sept. 2015), p. 26872. DOI: 10.1101/026872.

[54] Laleh Haghverdi et al. "Diffusion Pseudotime Robustly Reconstructs Lineage Branching". In: *Nature Methods* 13.10 (Oct. 2016), pp. 845–848. ISSN: 1548-7105. DOI: 10.1038/nmeth.3971.

[55] Manu Setty et al. "Wishbone Identifies Bifurcating Developmental Trajectories from Single-Cell Data". In: *Nat. Biotechnol.* 34.April (June 2016), pp. 1–14. ISSN: 1087-0156. DOI: 10.1038/nbt.3569.

[56] Cole Trapnell et al. "The Dynamics and Regulators of Cell Fate Decisions Are Revealed by Pseudotemporal Ordering of Single Cells." In: *Nature biotechnology* 32.4 (Mar. 2014), pp. 381–386. ISSN: 1546-1696. DOI: 10.1038/nbt.2859.

[57] Hirotaka Matsumoto and Hisanori Kiryu. "{{SCOUP}}: A Probabilistic Model Based on the {{Ornstein}}$\backslash
Process to Analyze Single-Cell Expression Data during Differentiation". In: *BMC Bioinformatics* 17 (June 2016), p. 232. ISSN: 1471-2105. DOI: 10.1186/s12859-016-1109-3.

[58] Xiaojie Qiu et al. "Reversed Graph Embedding Resolves Complex Single-Cell Trajectories". In: *Nature Methods* 14.10 (Oct. 2017), pp. 979–982. ISSN: 1548-7105. DOI: 10.1038/nmeth.4402.

[59] 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.

[60] Zhicheng Ji and Hongkai Ji. "{TSCAN}: Pseudo-Time Reconstruction and Evaluation in Single-Cell {RNA-Seq} Analysis". In: *Nucleic Acids Res.* (2016).

[61] Joshua D. Welch, Alexander J. Hartemink, and Jan F. Prins. "SLICER: Inferring Branched, Nonlinear Cellular Trajectories from Single Cell RNA-Seq Data". In: *Genome Biology* 17 (2016), p. 106. ISSN: 1474-760X. DOI: 10.1186/s13059-016-0975-3.

[62] David A DuVerle et al. "{{CellTree}}: An {{R}}/Bioconductor Package to Infer the Hierarchical Structure of Cell Populations from Single-Cell {{RNA}}-Seq Data". In: *BMC Bioinformatics* 17 (Sept. 2016), p. 363. ISSN: 1471-2105. DOI: 10.1186/s12859-016-1175-6.

[63] Tapio Lönnberg et al. "Single-Cell {{RNA}}-Seq and Computational Analysis Using Temporal Mixture Modeling Resolves {{TH1}}/{{TFH}} Fate Bifurcation in Malaria". In: *Science Immunology* 2.9 (Mar. 2017), eaal2192. ISSN: 2470-9468. DOI: 10.1126/sciimmunol.aal2192.

[64] Kieran R Campbell and Christopher Yau. "Probabilistic Modeling of Bifurcations in Single-Cell Gene Expression Data Using a {{Bayesian}} Mixture of Factor Analyzers". In: *Wellcome Open Research* 2 (Mar. 2017), p. 19. ISSN: 2398-502X. DOI: 10.12688/wellcomeopenres.11087.1.

[65] Luyi Tian et al. "{{scRNA}}-Seq Mixology: Towards Better Benchmarking of Single Cell {{RNA}}-Seq Protocols and Analysis Methods". In: *bioRxiv* (Oct. 2018), p. 433102. DOI: 10.1101/433102.

[66] 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.

[67] Valentine Svensson, Roser Vento-Tormo, and Sarah A Teichmann. "Exponential Scaling of Single-Cell {{RNA}}-Seq in the Past Decade". In: *Nature Protocols* 13.4 (Apr. 2018), pp. 599–604. ISSN: 1750-2799. DOI: 10.1038/nprot.2017.149.

[68] Junyue Cao et al. "Joint Profiling of Chromatin Accessibility and Gene Expression in Thousands of Single Cells". In: *Science* (Aug. 2018), eaau0730. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.aau0730.

[69] Natalya Pya and Simon N Wood. "Shape Constrained Additive Models". In: *Statistics and Computing* 25.3 (May 2015), pp. 543–559. ISSN: 1573-1375. DOI: 10.1007/s11222-013-9448-7.

[70] Morgan Taschuk and Greg Wilson. "Ten Simple Rules for Making Research Software More Robust". In: *PLOS Computational Biology* 13.4 (Apr. 2017), e1005412. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1005412.

[71] Serghei Mangul et al. "A Comprehensive Analysis of the Usability and Archival Stability of Omics Computational Tools and Resources". In: *bioRxiv* (Oct. 2018), p. 452532. DOI: 10.1101/452532.

[72] Greg Wilson et al. "Best {{Practices}} for {{Scientific Computing}}". In: *PLOS Biology* 12.1 (Jan. 2014), e1001745. ISSN: 1545-7885. DOI: 10.1371/journal.pbio.1001745.

[73] Haydee Artaza et al. "Top 10 Metrics for Life Science Software Good Practices". In: *F1000Research* 5 (Aug. 2016), p. 2000. ISSN: 2046-1402. DOI: 10.12688/f1000research.9206.1.

[74] Jeff Lee. *Rpackages: {{R}} Package Development - the {{Leek}} Group Way!* Dec. 2017.

[75] Hadley Wickham. *R Packages: Organize, Test, Document, and Share Your Code*. en. "O'Reilly Media, Inc.", Mar. 2015. ISBN: 978-1-4919-1056-6.

[76] Luis Bastiao Silva et al. "General Guidelines for Biomedical Software Development". In: *F1000Research* 6 (July 2017). ISSN: 2046-1402. DOI: 10.12688/f1000research.10750.2.

[77] Rafael C Jiménez et al. "Four Simple Recommendations to Encourage Best Practices in Research Software". In: *F1000Research* 6 (June 2017). ISSN: 2046-1402. DOI: 10.12688/f1000research.11407.1.

[78] Mehran Karimzadeh and Michael M Hoffman. "Top Considerations for Creating Bioinformatics Software Documentation". In: *Briefings in Bioinformatics* (). DOI: 10.1093/bib/bbw134.

[79] Alex Anderson. *Writing {{Great Scientific Code}}*. Oct. 2016.

[80] Brett K Beaulieu-Jones and Casey S Greene. "Reproducibility of Computational Workflows Is Automated Using Continuous Analysis". In: *Nature Biotechnology* 35.4 (Mar. 2017), nbt.3780. ISSN: 1546-1696. DOI: 10.1038/nbt.3780.

[81] Vincent Driessen. *A Successful {{Git}} Branching Model*. Jan. 2010.

[82] Anne-Laure Boulesteix. "Ten {{Simple Rules}} for {{Reducing Overoptimistic Reporting}} in {{Methodological Computational Research}}". In: *PLOS Computational Biology* 11.4 (Apr. 2015), e1004191. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1004191.

[83] Jean Francois Puget. *Green Dice Are Loaded (Welcome to p-Hacking)*. Mar. 2016.

[84] Frank Gannon. "The Essential Role of Peer Review". In: *EMBO Reports* 2.9 (Sept. 2001), p. 743. ISSN: 1469-221X. DOI: 10.1093/embo-reports/kve188.

[85] Melinda Baldwin. "In Referees We Trust?" In: *Physics Today* 70.2 (Feb. 2017), pp. 44–49. ISSN: 0031-9228. DOI: 10.1063/PT.3.3463.

[86] Mohamed Radhouene Aniba, Olivier Poch, and Julie D Thompson. "Issues in Bioinformatics Benchmarking: The Case Study of Multiple Sequence Alignment". In: *Nucleic Acids Research* 38.21 (Nov. 2010), pp. 7353–7363. ISSN: 0305-1048. DOI: 10.1093/nar/gkq625.

[87] Monika Jelizarow et al. "Over-Optimism in Bioinformatics: An Illustration". In: *Bioinformatics* 26.16 (Aug. 2010), pp. 1990–1998. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btq323.

[88] Wouter Saelens, Robrecht Cannoodt, and Yvan Saeys. "A Comprehensive Evaluation of Module Detection Methods for Gene Expression Data". In: *Nature Communications* 9.1 (Mar. 2018), p. 1090. ISSN: 2041-1723. DOI: 10.1038/s41467-018-03424-4.

[89] Gioele La Manno et al. "{{RNA}} Velocity of Single Cells". In: *Nature* 560.7719 (Aug. 2018), pp. 494–498. ISSN: 1476-4687. DOI: 10.1038/s41586-018-0414-6.

[90] Raquel Norel, John Jeremy Rice, and Gustavo Stolovitzky. "The Self-Assessment Trap: Can We All Be Better than Average?" In: *Molecular systems biology* 7.1 (2011), p. 537. ISSN: 1744-4292. DOI: 10.1038/msb.2011.70.

[91] Anthony Gitter. *Single-Cell RNA-Seq Pseudotime Estimation Algorithm*. https://github.com/agitter/single-cell-pseudotime. June 2018. DOI: 10.5281/zenodo.1297423.

[92] Tsukasa Kouno et al. "Temporal Dynamics and Transcriptional Control Using Single-Cell Gene Expression Analysis". In: *Genome Biol.* 14.10 (2013), R118.

[93] Chun Zeng et al. "Pseudotemporal Ordering of Single Cells Reveals Metabolic Control of Postnatal β Cell Proliferation". In: *Cell Metabolism* 25.5 (May 2017), 1160–1175.e11. ISSN: 15504131. DOI: 10.1016/j.cmet.2017.04.014.

[94] 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.

[95] 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.

[96] 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.

[97] 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.

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

[99] Daniel Marbach et al. "Tissue-Specific Regulatory Circuits Reveal Variable Modular Perturbations across Complex Diseases". In: *Nature Methods* 13.4 (Apr. 2016), p. 366. ISSN: 1548-7105. DOI: 10.1038/nmeth.3799.

[100] Toni Giorgino. "Computing and Visualizing Dynamic Time Warping Alignments in R: The Dtw Package". In: *Journal of Statistical Software* 7 (Sept. 2009). DOI: 10.18637/jss.v031.i07.

[101] Paolo Tormene et al. "Matching Incomplete Time Series with Dynamic Time Warping: An Algorithm and an Application to Post-Stroke Rehabilitation". In: *Artificial Intelligence in Medicine* 45.1 (Jan. 2009), pp. 11–34. ISSN: 0933-3657. DOI: 10.1016/j.artmed.2008.11.007.

[102] Aaron T L Lun, Davis J McCarthy, and John C Marioni. "A Step-by-Step Workflow for Low-Level Analysis of Single-Cell {{RNA}}-Seq Data with {{Bioconductor}}". In: *F1000Research* 5 (Oct. 2016), p. 2122. ISSN: 2046-1402. DOI: 10.12688/f1000research.9501.2.

[103] G Jurman et al. "The {{HIM}} Glocal Metric and Kernel for Network Comparison and Classification". In: *2015 {{IEEE International Conference}} on {{Data Science}} and {{Advanced Analytics}} ({{DSAA}})*. Oct. 2015, pp. 1–10. DOI: 10.1109/DSAA.2015.7344816.

[104] Marvin N Wright and Andreas Ziegler. "Ranger: {{A Fast Implementation}} of {{Random Forests}} for {{High Dimensional Data}} in {{C}}++ and {{R}} | {{Wright}} | {{Journal}} of {{Statistical Software}}". In: *Journal of Statistical Software* 77.1 (Mar. 2017). DOI: 10.18637/jss.v077.i01.

[105] T Junttila and P Kaski. "Engineering an {{Efficient Canonical Labeling Tool}} for {{Large}} and {{Sparse Graphs}}". In: *2007 {{Proceedings}} of the {{Ninth Workshop}} on {{Algorithm Engineering}} and {{Experiments}} ({{ALENEX}})*. Proceedings. Society for Industrial and Applied Mathematics, Jan. 2007, pp. 135–149. DOI: 10.1137/1.9781611972870.13.

[106] Laura Bahiense et al. "The Maximum Common Edge Subgraph Problem: {{A}} Polyhedral Investigation". In: *Discrete Applied Mathematics*. V Latin American Algorithms, Graphs, and Optimization Symposium $\backslash$textemdash{} Gramado, Brazil, 2009 160.18 (Dec. 2012), pp. 2523–2541. ISSN: 0166-218X. DOI: 10.1016/j.dam.2012.01.026.

[107] Edward R Dougherty. "Validation of Gene Regulatory Networks: Scientific and Inferential". In: *Briefings in Bioinformatics* 12.3 (May 2011), pp. 245–252. ISSN: 1477-4054. DOI: 10.1093/bib/bbq078.

[108] Mads Ipsen and Alexander S Mikhailov. "Evolutionary Reconstruction of Networks". In: *Physical Review. E, Statistical, Nonlinear, and Soft Matter Physics* 66.4 Pt 2 (Oct. 2002), p. 46109. ISSN: 1539-3755. DOI: 10.1103/PhysRevE.66.046109.

[109] R. Albert. "Network Inference, Analysis, and Modeling in Systems Biology". In: *the Plant Cell Online* 19.11 (2007), pp. 3327–3338. ISSN: 1040-4651. DOI: 10.1105/tpc.107.054700.

[110] Varun Narendra et al. "A Comprehensive Assessment of Methods for De-Novo Reverse-Engineering of Genome-Scale Regulatory Networks". In: *Genomics* 97.1 (2011), pp. 7–18. ISSN: 08887543. DOI: 10.1016/j.ygeno.2010.10.003.

[111] Tarmo Äijö and Richard Bonneau. "Biophysically Motivated Regulatory Network Inference: Progress and Prospects". In: *Human Heredity* 81.2 (2017), pp. 62–77. ISSN: 14230062. DOI: 10.1159/000446614.

[112] Fabrício M. Lopes et al. "A Feature Selection Technique for Inference of Graphs from Their Known Topological Properties: Revealing Scale-Free Gene Regulatory Networks". In: *Information Sciences* 272 (2014), pp. 1–15. ISSN: 00200255. DOI: 10.1016/j.ins.2014.02.096.

[113] Joeri Ruyssinck et al. "Netter: Re-Ranking Gene Network Inference Predictions Using Structural Network Properties." In: *BMC Bioinformatics* 17.1 (2016), p. 76. ISSN: 1471-2105. DOI: 10.1186/s12859-016-0913-0.

[114] Alexander W Rives and Timothy Galitski. "Modular Organization of Cellular Networks." In: *Proceedings of the National Academy of Sciences of the United States of America* 100.3 (2003), pp. 1128–33. ISSN: 0027-8424. DOI: 10.1073/pnas.0237338100.

[115] L H Hartwell et al. "From Molecular to Modular Cell Biology." In: *Nature* 402.6761 Suppl (1999), pp. C47–C52. ISSN: 0028-0836. DOI: 10.1038/35011540.

[116] a Barabasi et al. "Network Biology: Understanding the Cell's Functional Organization." In: *Nature reviews. Genetics* 5.2 (Feb. 2004), pp. 101–13. ISSN: 1471-0056. DOI: 10.1038/nrg1272.

[117] R Milo et al. "Network Motifs: Simple Building Blocks of Complex Networks." In: *Science (New York, N.Y.)* 298.2002 (2002), pp. 824–827. ISSN: 00368075. DOI: 10.1126/science.298.5594.824.

[118] N Przulj et al. "Modeling Interactome: Scale-Free or Geometric?" In: *Bioinformatics (Oxford, England)* 20.18 (Dec. 2004), pp. 3508–15. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bth436.

[119] Tijana Milenković, Nataša Pržulj, and Natasa Przulj. "Uncovering Biological Network Function via Graphlet Degree Signatures." In: *Cancer informatics* 6 (Jan. 2008), pp. 257–73. ISSN: 1176-9351.

[120] Cortnie Guerrero et al. "Characterization of the Proteasome Interaction Network Using a QTAX-Based Tag-Team Strategy and Protein Interaction Network Analysis." In: *Proceedings of the National Academy of Sciences of the United States of America* 105.36 (2008), pp. 13333–13338. ISSN: 0027-8424. DOI: 10.1073/pnas.0801870105.

[121] Omkar Singh, Kunal Sawariya, and Polamarasetty Aparoy. "Graphlet Signature-Based Scoring Method to Estimate Protein-Ligand Binding Affinity." In: *Royal Society open science* 1.4 (2014), p. 140306. ISSN: 2054-5703. DOI: 10.1098/rsos.140306.

[122] Tijana Milenković et al. "Optimal Network Alignment with Graphlet Degree Vectors". In: *Cancer informatics* (2010), pp. 121–137.

[123] Oleksii Kuchaiev et al. "Topological Network Alignment Uncovers Biological Function and Phylogeny." In: *Journal of the Royal Society, Interface / the Royal Society* 7.50 (2010), pp. 1341–1354. ISSN: 1742-5662. DOI: 10.1098/rsif.2010.0063.

[124] T Milenković, H Zhao, and FE Faisal. "Global Network Alignment in the Context of Aging". In: *Proceedings of the International Conference on Bioinformatics, Computational Biology and Biomedical Informatics* (2013), pp. 23–32.

[125] Nino Shervashidze et al. "Efficient Graphlet Kernels for Large Graph Comparison". In: *AISTATS* 5 (2009), pp. 488–495.

[126] Vladimir Vacic et al. "Graphlet Kernels for Prediction of Functional Residues in Protein Structures." In: *Journal of computational biology : a journal of computational molecular cell biology* 17.1 (2010), pp. 55–72. ISSN: 1557-8666. DOI: 10.1089/cmb.2009.0029.

[127] David Eppstein and Emma S. Spiro. "The H-Index of a Graph and Its Application to Dynamic Subgraph Statistics". In: *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 5664 LNCS. Springer-Verlag, 2009, pp. 278–289. ISBN: 3-642-03366-0. DOI: 10.1007/978-3-642-03367-4_25.

[128] David Eppstein et al. "Extended Dynamic Subgraph Statistics Using H-Index Parameterized Data Structures". In: *Theoretical Computer Science* 447 (Aug. 2012), pp. 44–52. ISSN: 03043975. DOI: 10.1016/j.tcs.2011.11.034.

[129] J E Hirsch. "An Index to Quantify an Individual's Scientific Research Output". In: *Proceedings of the National Academy of Sciences of the United States of America* 102.46 (Nov. 2005), pp. 16569–72. ISSN: 0027-8424. DOI: 10.1073/pnas.0507655102.

[130] Tomaž Hočevar and Janez Demšar. "A Combinatorial Approach to Graphlet Counting." In: *Bioinformatics (Oxford, England)* 30.4 (Feb. 2014), pp. 559–65. ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btt717.

[131] Réka Albert and Albert Laszlo Barabasi. "Statistical Mechanics of Complex Networks". In: *Reviews of Modern Physics* 74.January (2002), pp. 47–97. ISSN: 1478-3967. DOI: 10.1088/1478-3967/1/3/006.

[132] P. Erdős and A Rényi. "On Random Graphs". In: *Publicationes Mathematicae* 6 (1959), pp. 290–297. ISSN: 00029947. DOI: 10.2307/1999405.

[133] M J B Appel and R P Russo. "The Minimum Vertex Degree of a Graph on Uniform Points in [0,1]d". In: *Adv. in Appl. Probab.* 29.3 (1997), pp. 582–594. ISSN: 00018678.

[134] Vân Anh Huynh-Thu et al. "Inferring Regulatory Networks from Expression Data Using Tree-Based Methods". In: *PLoS ONE* 5.9 (Jan. 2010), e12776. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0012776.

[135] Marco Moretto et al. "COLOMBOS v3.0: Leveraging Gene Expression Compendia for Cross-Species Analyses". In: *Nucleic Acids Research* 44.D1 (2016), pp. D620–D623. ISSN: 13624962. DOI: 10.1093/nar/gkv1251.

[136] R. Edgar. "Gene Expression Omnibus: NCBI Gene Expression and Hybridization Array Data Repository". In: *Nucleic Acids Research* 30.1 (2002), pp. 207–210. ISSN: 13624962. DOI: 10.1093/nar/30.1.207.

[137] Socorro Gama-Castro et al. "RegulonDB Version 9.0: High-Level Integration of Gene Regulation, Coexpression, Motif Clustering and Beyond". In: *Nucleic Acids Research* 44.D1 (2016), pp. D133–D143. ISSN: 13624962. DOI: 10.1093/nar/gkv1156.

[138] Sisi Ma et al. "De-Novo Learning of Genome-Scale Regulatory Networks in S. Cerevisiae". In: *PLoS ONE* 9.9 (2014). ISSN: 19326203. DOI: 10.1371/journal.pone.0106479.