Developing and benchmarking methods for analysing transcriptomics data

Ontwikkelen en vergelijken van methoden voor transcriptoom data-analyse

Wouter Saelens

https://orcid.org/0000-0002-7114-6248
https://github.com/zouter

Supervisors: prof. dr. Yvan Saeys, prof. dr. Bart Lambrecht
Dissertation submitted in fulfillment of the requirements for the degree of
Doctor of Science: Bioinformatics

GHENT UNIVERSITY

Department of Applied Mathematics, Computer science and Statistics
VIB - UGent Center for Inflammation Research
Faculty of Sciences
Ghent University

Part of the work in this dissertation was conducted within the Faculty of Sciences at Ghent University. This work was also supported by a Bijzonder onderzoeksfonds grant from Ghent University.

Part of the work in this dissertation was conducted within the VIB-UGent Center for Inflammation Research.

Part of the work in this dissertation was conducted during visit to Institute of Molecular Life Sciences at University of Zürich.

This work was supported by an aspirant grant and a travel grant from Fonds Wetenschappelijk Onderzoek.

Samenvatting

Het bepalen van de hoeveelheden messenger RNA (mRNA) moleculen die aanwezig zijn in een cel laat ons toe om beter een cel zijn functie en ontwikkeling te karakteriseren. Deze transcriptoom data kan dan op verschillende manieren worden geanalyseerd, bijvoorbeeld door op zoek te gaan naar gelijkaardig geëxpresseerde genen, of door het vinden van differentiatie paden die cellen nemen tijdens hun ontwikkeling.

In deze thesis presenteer ik drie van mijn bijdragen voor de analyse van transcriptoom data. Allereerst beschrijf ik een methode die het gemakkelijker maakt om het transcriptoom van drie biologische stalen te onderzoeken. Ik gebruik deze methode om nieuwe inzichten te krijgen in de ontwikkeling van macrofagen, een type immuuncel. Ik voer daarna een vergelijking uit tussen verschillende methoden om gelijkaardige geëxpresseerde genen te halen uit transcriptoom data. We hebben hierbij gevonden dat bepaalde methoden beter presenteren dan de populaire clusteringsmethoden, alhoewel vernieuwingen op het vlak van interpretatie noodzakelijk zijn om deze gebruiksvriendelijker te maken. In een laatste bijdrage presenteer ik een vergelijking van methoden om cellulaire differentiatie paden te achterhalen. Deze methoden zijn nog maar zeer recent geïntroduceerd, hand in hand met de ontwikkeling van technologie die individuele cellen kan analyseren. Ik beschrijf de nauwkeurigheid, schaalbaarheid, stabiliteit en bruikbaarheid van 45 methoden op echt en synthetische data. Op basis van deze informatie hebben we een reeks van context-afhankelijke richtlijnen opgesteld for gebruikers. We wijzen ook op verschillende uitdagingen die in toekomstige methoden zullen moeten worden opgelost.

Verder bouwend op deze laatste bijdrage, geef ik twee mogelijkheden voor verder onderzoek. Ik beschrijf eerst een set van tools die gebruikt maakt van onze vergelijking om het gemakkelijker voor gebruikers te maken om paden in single-cell data te vinden. Telkens vermeld ik echter ook verschillende nieuwe methoden die ons meer inzichten zullen kunnen verschaffen in deze cellulaire paden. Deze zullen ons in de toekomst kunnen helpen om ziektes te bestuderen die veroorzaakt zijn door problemen in cellulaire ontwikkeling. Ik besluit met enkele ideëen voor betere vergelijkingen van methoden binnen de computationele biologie. Deze zullen het toelaten dat onderzoekers beter samenwerken terwijl ze een methode ontwikkelen en vergelijken, zodat het steeds duidelijk is welke methode het best geschikt is om een biomedische dataset te analyseren.

Summary

Profiling which messenger RNA (mRNA) molecules are expressed can give us important functional and developmental information about a cell. Computational methods are necessary to analyse the large amounts of information present in such a transcriptome, for example by grouping similar genes into functional modules, or by finding trajectories in the data.

In this thesis, I present three of my contributions to the analysis of transcriptomics data. I first describe a method that makes it easier to visualise and interpret the transcriptome of three biological samples. I showcase an application of this method on transcriptomics data from macrophage progenitors. I then present a benchmark for methods that group genes into co-expressed modules. In this benchmark study, we found that decomposition techniques can outperform the more popular clustering methods, although some improvements on interpretation and visualisation may be required before they can be broadly useful. In a final contribution, I present a benchmark of methods that can delineate paths that cells take during development or activation. These trajectory inference methods have only been very recently introduced, hand in hand with technological improvements in single-cell sequencing. I describe a comprehensive benchmark of the accuracy, scalability, stability and usability of 45 methods on real and synthetic data. We develop a set of context-dependent guidelines for method users, and pinpoint several challenges which need to be addressed in future methods.

Building further upon this last contribution, I outline two exciting avenues for future research. I describe a toolkit that incorporates the benchmark results to make trajectory inference more streamlined for the end user. To make this toolkit more useful, I indicate several opportunities for new visualisation and interpretation methods. These will in the end allow us to get a detailed functional insight into developmental diseases. Finally, I also present a roadmap to improve the way benchmarking is done in computational biology, by incorporating several tools of modern software development. These will make method evaluations more collaborative and continuous, so that it is clear at any time which method is best suited to study human diseases.

Dankwoord (*Acknowledgments*)

In deze thesis sta ik op de schouders van reuzen. Reuzen die mij kansen hebben geboden om aan onderzoek te doen. Reuzen die mij hebben ondersteund terwijl het onderzoek toch niet ging zoals verwacht. En vooral reuzen die een kleine microbe in mij hebben gestoken die dag en nacht alles wil onderzoeken, zelfs als de zon schijnt of België een goed WK meemaakt.

De voornaamste figuurlijke reus is natuurlijk mijn promoter, Yvan. Het is allemaal begonnen toen hij daar stond als een enthousiaste biomedische post-doc tussen een overwegend planten bio-informatica bij de voorstelling van master thesis projecten. Zes jaar later is hij uitgegroeid tot een vooraanstaande professor in machine learning en (single-cell) bio-informatica. Doorheen die jaren ben je mij steeds mooi blijven steunen in mijn projecten. Ik stel het heel hard op prijs dat je mijn samenwerkingen met Robrecht, Liesbet en Robin hebt bevorderd. Ik heb een vermoeden dat ik vaak wat eigenzinnig was in jouw ogen, maar ik hoop dat je toch tevreden bent over het resultaat! Ik heb ook ontzettend veel geleerd van mijn co-promoter Bart, ik sta nog steeds versteld van hoe hij zijn brede biologische expertise kan toepassen die hij telkens toonde tijdens de monday morning meetings.

Die andere grote reus is ongetwijfeld Robrecht. Met onze verschillende achtergronden hebben we elkaar al heel vroeg kunnen bij staan met allerhande vragen. Uiteindelijk vatten we het idee op om een groot gezamelijk project op te zetten. Eerst begon dit wat aftastend, elk met onze eigen github repository en met minimale ”bemoeienis”. Naarmate het vorderde werd de samenwerking intenser, maar wonder boven wonder bleek die ook goed te lukken! Ik vond het prachtig hoe we elkaars ideeën in de praktijk konden omzetten; wanneer één persoon een goed idee had maar de moed niet kon vinden om het te implementeren, was de andere al na enkele dagen klaar met een prototype. Ik heb heel veel geleerd van jou: hard-skills zoals properdere code en code structureren, maar ook soft-skills zoals samenwerken, en leren op de tong te bijten wanneer ik het weeral niet eens was met jou idee. We hebben samen iets mooi uit de grond gestampt!

Liesbet, je bent voor mij een reuzin op vele vlakken. Al vanaf mijn masterthesis stond je altijd klaar om mijn vragen te beantwoorden rond het analyseren van transcriptoom data. Als je zelf dan met wat vragen zat, probeerde ik je ook zo goed mogelijk te helpen. En daar zijn meermaals mooie dingen uit voort gekomen. Je bent voor mij een groot voorbeeld van een nauwe samenwerking tussen de computationele en experimentele biologie!

Sofie, het was altijd heel leuk om samen fouten in code op te lossen, figuren te ontrafelen, te schilderen en over ons onderzoek en daarbuiten te babbelen. Toen je een suggestie gaf over een bepaalde figuur was ik vaak eerst wat sceptisch, maar toen ik

het effectief toepastte was ik wel altijd heel blij met het resultaat! Robin, mooi hoe je een idee waar ik al even mee speelde omzette naar een volledig onderzoeksproject! Isaac, it was great to discuss machine learning problems with you. If you're still wondering: yes, I finally submitted the paper, and it was accepted. And I really like your computerphile video, and I hereby agree with the comment of fellow youtuber hcblue: "Oh man, more of Dr. Triguero please! :D". Rest assured, your legacy continues in the form of... Dani. You are really great at explaining complex machine learning stuff to me. There are so many other giants to thank (no wonder that we urgently needed a new office): Paco, Helena, Ruth, Annelies, Sarah, Pieter, Maxim, Quentin, Robin, Joris, Kevin, Katrien and Jonathan.

I was also lucky to stand on the shoulder of many biological giants. Martin, you're honest feedback on our algorithms is really important to push our methods to the limit! Charlie, great to have worked and pondered on so many of your datasets. Lianne, thank you being patient when we were finalising the algorithm! And of course all the others in Bart's, Charlie's and Martin's groups: you are all awesome!

A *giant* thank you to all the people in Zürich as well. Mark, I really liked working in your group. And actions say more than words of course, so I'm secretly hoping to be able to work with your group later on in my career as well! Lukas and Charlotte, you were my two biggest examples when it comes down to benchmarking, so while working with you I felt like a beginning actor working with Leonardo Dicaprio. And thanks to Paolo, Iza, Simone, Almut, Stephan, Katharina, Pierre-Luc, Helena, Fiona and Stephany, for such a warm and welcome stay in Switzerland!

Dankjewel mama en papa om mijn wetenschappelijke curiositeit altijd te blijven ondersteunen! Ik denk dat die investering in onder andere een kinder encyclopedie wel heeft opgebracht. Ook mijn schoonouders hebben mij bij de laatste loodjes heel mooi ondersteunt. Alleen de schoongrootvaders hebben het wat lastig gemaakt om te blijven doctoreren. Want de passie waarmee zij nog steeds aan houtbewerking doen: je zou bijna je doctoraat opgeven en schrijnwerker beginnen worden...

Aan mijn drie zussen ten slotte, Marieke, Anneleen en Veerle, ook een dikke merci. Ik denk niet dat het toevallig is dat jullie met drie waren, want het is gelijk aan het aantal reviewers dat je krijgt na submissie van een paper. All gekheid op een stokje, ik heb heel veel aan jullie gehad!

Wouter Saelens

20 juni 2019

Copyright

List of abbreviations

AM	Alveolar macrophage
BM	Bone marrow
CD	Cluster of differentiation
cDC	Conventional dendritic cell
cDNA	Complementary deoxyribonucleic acid
DC	Dendritic cell
FACS	Fluorescence assisted cell sorting
FL	Fetal liver
IFNB1	Interferon beta 1
mRNA	Messenger ribonucleic acid
MO	Monocyte
NI	Network inference
PCA	Principal component analysis
pre-DC	Precursor dendritic cell
RNA-seq	Ribonucleic acid sequencing
RMA	Robust multi-array average
TF	Transcription factor
TI	Trajectory inference
UMI	Unique molecular identifier
XCR1	X-C motif chemokine receptor 1
YS	Yolk sac

Table of Contents

1 Introduction

A large part of a cell's identity is driven by the molecules it contains: the lipids, proteins, nucleic acids and metabolites all determine how a cell fulfils its function, and how it reacts to its environment. By determining the composition and amounts of these molecules, it is therefore possible to better understand the inner workings of a cell and its impact on the organism and ecosystem. This hypothesis has been the main driving force behind -omics technologies, which try to profile and quantify these molecules within (populations of) cells.

The focus within my thesis primarily lies on one kind of molecules: messenger RNAs (mRNA), and conversely on the field of transcriptomics. mRNAs are interesting to study because they contain the information that is necessary to create proteins, which are the main - albeit not the only - molecules that fulfil internal and external cellular functions. To produce a particular protein, a cell first needs to produce a certain mRNA. The amount of proteins in a cell is often driven by the amount of corresponding mRNAs, which therefore serves as some kind proxy for protein abundance within this cell [1]. Another reason to study mRNAs is simply because they are relatively easy and less expensive to profile, at least compared to the more functionally relevant proteins.

In this thesis, I use transcriptome to denote all mRNA molecules in a cell. It should be noted however that there are many other RNA molecules within a cell that also have a functional importance, although they do not encode for a protein.

mRNA production and gene regulation

The recipe for a particular mRNA molecule is contained in the genome of a cell, within stretches of DNA called genes. Around and within the gene are regions of DNA that contain information on how the gene is regulated. These DNA sequences are detected by transcription factors (TF), which will influence the speed by which the nearby genes are used as template for an mRNA molecule, within a process called transcription. In this way, transcription factors are thought to be the main drivers behind variation in gene expression within and between organisms, although many other post-transcriptional processes also play a role.

In prokaryotes, TFs mainly bind in the promoter region, a stretch of DNA that comes right before the position where transcription starts. Together with specific sequences in the promoter region, they help or prevent the recruitment of the RNA polymerase complex to the promoter. TFs interact with each other and other molecules, such as metabolites, and their activity can therefore be context-dependent. Apart from transcription factors, some prokaryotic genes have other means by which transcripts is regulated, for example through riboswitches.

Transcriptional regulation is more complex within eukaryotes. The DNA is packed together with histone proteins into a high-density structure called the chromatin. Packaging of the DNA often limits the accessibility to the RNA polymerase and of other factors necessary for transcription [2]. Histone proteins can undergo post-translational modifications which may enhance or limit the accessibility of DNA. These modifications are in turn be regulated by transcription factors, through activation or inhibition of chromatin remodelling enzymes [3]. Histone modifications can also influence the cycle of transcription, through regulation of the initiation, RNA polymerase release , elongation and termination phases [4].

When the DNA is accessible, transcription factors assemble the pre-initiation complex, which contain general transcription factors that in the end recruit the RNA polymerase. While these transcription factors may bind sequences close to the initiation site (such as the TATA box), some TFs may bind very far away from the promoter sequence [5]. Here, the presence of long-range enhancer-promoter interactions is driven by the three-dimensional organisation of the chromosomes [6].

Transcription initiation is only the first step in the life of an mRNA, and many other processes may further influence its abundance. The RNA polymerase will first create a pre-mRNA molecule, which has to be processed in three ways: capping of the 5' terminus, splicing of introns and polyadenylation. All these processed can be impacted by co-factors, which in turn may affect the stability and localisation of the resulting mRNA [7]. For example, a short poly-A tail can enhance the decay of an mRNA, although its length is not necessarily a major determinant in mRNA abundance [8].

The mRNA decay rate and translational efficiency can be further regulated by both proteins and other RNAs in the cytosol. The latter occurs in the form of micro-RNAs and small interfering RNAs, which are short RNA molecules with partial complementarity to their target mRNA and which help the recruitment of an RNA induced silencing complex to the transcript [9].

It is important to note that although the regulation of certain genes is well studied (for example the human IFNB1 gene), for most genes a detailed model of regulation is not available. It is therefore still very difficult to predict how perturbations in transcription factors will influence the activity of a gene. Although a lot is known about gene regulation, it is also clear that many things are still unknown [6].

Technologies to determine the transcriptome

Because mRNAs are very unstable after cell lysis and typically only have a couple of dozen copies per cell [10], sensitivity is perhaps one of the most challenging parts of determining the transcriptome [11, 12]. These issues can at least be partially overcome by first converting mRNAs into the much more stable DNA, and then amplifying it using a polymerase chain reaction. The mRNAs are often enriched by using a poly-T primer, which hybridises RNA molecules that contain a poly-A tail. These complementary DNAs (cDNAs) can then be profiled by making use of their natural tendency to bind with DNA probes, or by sequencing and counting.

I first briefly discuss techniques to study the transcriptome of many cells (”bulk” transcriptomics), after which I will delve into miniaturised versions which work on the level of single cells.

Determining the transcriptome of many cells

After some initial technologies using expressed sequence tags and serial analysis of gene expression, the two technologies that really established the field of transcriptomics were microarrays and RNA-sequencing (RNA-Seq).

Microarrays make use of the tendency of DNA to form a strong double-stranded helix with a complementary DNA molecule. Pieces of cDNA are first labelled using fluorochromes, and are then brought onto a plate in which DNA probes are fixed at certain spots. After a washing step, the hybridised cDNA can be easily quantified by looking at the fluorescence intensities of the individual spots. Microarrays were the most sensitive and cost-efficient technology to measure all the mRNAs within a sample during the 2000s. They really established transcriptomics analyses as an important technique to study biological systems. This is exemplified by the over 1 million samples that have been submitted to the gene expression omnibus [13].

Despite its popularity, microarray analyses have several limitations, such as the bias introduced by selecting the probes, the need for a reference genome (or transcriptome) and the relatively low sensitivity. RNA-seq overcomes these limitations and has therefore in the recent years become the standard for determining the transcriptome [13]. Because it is a de novo transcriptomics technique, it is not biased towards particular probes nor does it require a reference, although one still has to take into account biases introduced by purification and sequencing [14]. Some other advantages of RNA-seq include the improved ability to delineate alternative splicing, additional information on genomic differences [15, 16] and the ability to determine the transcriptome of complex microbial communities [17].

The technology keeps on advancing. Miniaturization combined with innovations in sequencing further decreases the cost of sequencing the transcriptome [18]. Moreover, new techniques such as long-read RNA sequencing finally makes it possible to unambiguously assess mRNA isoform usage [19].

Determining the transcriptome of a single cell

Although RNA-seq provides an cost-effective way to determine the transcriptome of a population of cells, such an analysis can easily hide the actual transcriptomic heterogeneity in the sample. Even when cells are phenotypically the same, there is no guarantee that they are transcriptionally (and functionally) homogeneous. Furthermore, often we're interested in only a small subset of cells, e.g. stem cells or cells responding to a stimulus. This highlights the importance of scaling -omics analyses to individual cells, but introduces two main technical hurdles: labelling of mRNAs of the same cells, and quantification of the tiny amounts of mRNAs in a sensitive way. In essence, single-cell transcriptomics methods are thus extensions of bulk RNA-seq methods, but with extra mRNA labelling and amplification steps.

Many techniques exist to label individual cells, and this often involves single-cell isolation and subsequent addition of cell barcodes to the cDNAs. Isolation of single-cells can be done by using manual picking [20], integrated fluidic circuits [21], cell sorting into microwells [22, 23], random separation into droplets [24, 25], or random separation into picowells [26, 27]. Recent technologies skip the isolation step altogether, by first fixing the cells and performing library preparation *in situ*. In such cases, cells are labelled by either many rounds of combinatorial barcoding [28], or a transfer onto a barcoded plate [29]. Single-cell labelling is the major determinant of the cell throughput of a particular platform, and also strongly contributes to its cost-effectiveness. A couple of years ago the fluidigm C1 (integrated fluidic circuits) and Smart-Seq2 (cell sorting) systems were the most popular ways of doing single-cell transcriptomics, and were used to generate datasets of a couple of hundred up to a few thousands of cells. Now, these have been largely superseded by droplet-based techniques such as Drop-Seq and the Chromium system from 10X genomics, which allow scaling up to a million of cells. Perhaps these in turn will soon be superseded by *in situ* technologies that don't need any isolation, as it is this process that can easily induce a cell selection bias in the analysis towards relatively small cells with a regular shape [29].

Even when mRNAs or cDNAs are isolated, it is still challenging to quantify them because of the extremely small amounts of RNA present. All technologies therefore first amplify the cDNA, for which there are two popular strategies. One uses PCR amplification by either ligating a poly-A tail to the 3' end of the transcripts [20] or using template-switching adapter [30]. The other uses *in vitro* transcription to produce many mRNA copies of the same transcript, after which these can be again converted back to cDNA [31, 27].

When individual cDNAs are barcoded and amplified, the cDNAs can be brought together and sequenced as in the bulk methods.

Many other omics technologies can also be transferred to the single-cell level, although sensitivity is often the limiting factor [32]. This includes the genome [33], chromatin accessibility [34] and the proteome [35]. What is particularly of interest are techniques that can assess multiple modalities at once, such as the transcriptome combined with chromatin accessibility or protein expression (Figure 1.1). These techniques typically try to convert each molecule into a "common molecular format", such as a cDNA molecule [36]. At the moment, many of these techniques require some extensive technical expertise, and commercialisation or open-sourcing [37] will be necessary to make them commonplace in biological studies.

Computational methods to analyse the transcriptome

Transcriptomics data is analysed using a variety of methods, mostly adapted from machine learning and statistics, with some adaptations to better match the characteristics of the data. Rather then delving too much into technical details, I will focus on the main ideas, why they're necessary, and what biological conclusions we can make by using them. Most of the methods that I discuss here will in some form be used in the methods that are described in the other chapters of this thesis. In each case I discuss both bulk and single-cell tools intertwined. Although almost everyone uses different tools for the two data types [40], the core concepts behind every type of method remain largely the same. An overview of the different kinds of analyses along with the most prominent methods is given in Figure 1.2.

Preprocessing and normalisation

We want to use transcriptomics data to make biological conclusions, not technical ones. But often, the variation in a real dataset is driven by some unwanted technical (such as temperature, sequencing depth) or biological (stress response, cell death) differences between samples. While some of these issues can be circumvented by good experimental design, the unwanted sources of variation that are still present

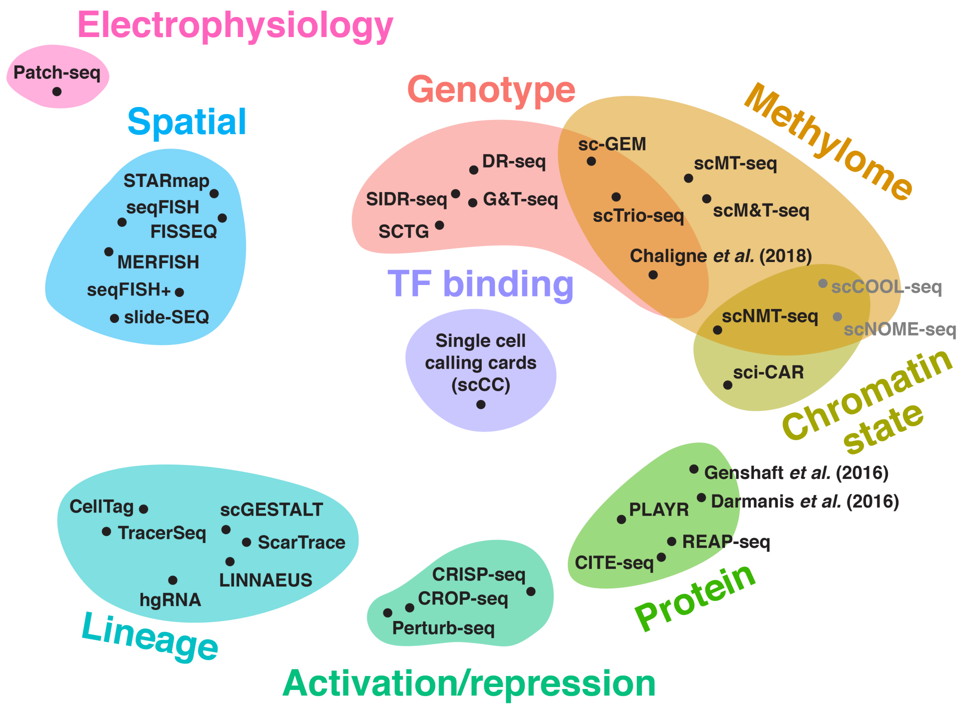
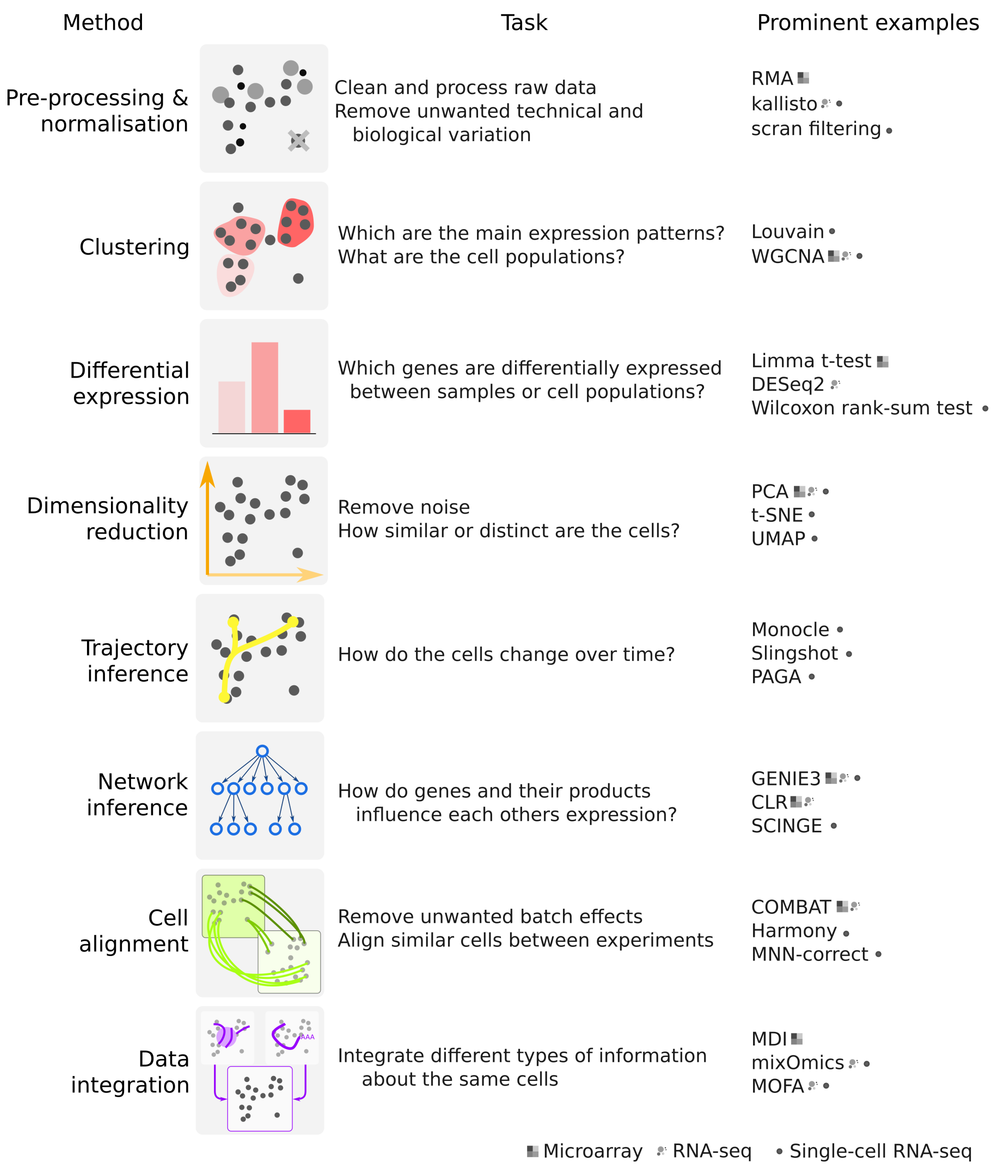


Figure 1.1: Single-cell techniques that can profile other modalities next to mRNA levels. Two dimensional positioning was done manually based on similarity between the different modalities. Figure adapted from [38] with some recent additions (Licence: CC BY-NC-SA). When converting each modality to a common ”molecular format”, the possibilities become endless. This is demonstrated by one recent technology that combines five different modalities [39], which would be hard to visualise on a chart like this.



Method	Task	Prominent examples
Pre-processing & normalisation	Clean and process raw data Remove unwanted technical and biological variation	RMA ■ kallisto ⁘ ● scran filtering ●
Clustering	Which are the main expression patterns? What are the cell populations?	Louvain ● WGCNA ■ ⁘ ●
Differential expression	Which genes are differentially expressed between samples or cell populations?	Limma t-test ■ DESeq2 ⁘ Wilcoxon rank-sum test ●
Dimensionality reduction	Remove noise How similar or distinct are the cells?	PCA ■ ⁘ ● t-SNE ● UMAP ●
Trajectory inference	How do the cells change over time?	Monocle ● Slingshot ● PAGA ●
Network inference	How do genes and their products influence each others expression?	GENIE3 ■ ⁘ ● CLR ■ ⁘ SCINGE ●
Cell alignment	Remove unwanted batch effects Align similar cells between experiments	COMBAT ■ ⁘ Harmony ● MNN-correct ●
Data integration	Integrate different types of information about the same cells	MDI ■ mixOmics ⁘ ● MOFA ⁘ ●

■ Microarray ⁘ RNA-seq ● Single-cell RNA-seq

Figure 1.2: Overview of methods used to analyse transcriptome data. For each type of method, the most prominent examples are given along with for which kind technology they are most often used (although they are rarely restricted to that technology): ■ microarray, ⁘ RNA-sequencing, ● Single-cell RNA-sequencing

should be characterised computationally, and in some cases removed, before the data is ready to be interpreted biologically.

Given its status as very mature technology, preprocessing and normalisation of most microarrays is done by the *de facto* standard Robust Multichip Average (RMA) method [41]. This method will first try to remove spatial differences between probe expression on the same array. After a log transformation, the differences between arrays are normalised using quantile normalisation, which will make the distribution between all arrays the same. This step makes the assumption that each condition has about the same number of genes up- as down-regulated, which can be easily violated depending on system under study [42].

Preprocessing of RNA-seq data is a bit more complicated, given that the output after sequencing are short cDNA sequences. The first step is to filter and trim the raw reads so that they contain high quality reads with no adapter sequences. The next steps depend on the use case. If the main goal is quantification of gene expression, alignment-free methods are the fastest option [43, 44], taking into account some of its limitations [45]. A greater level of detail can be obtained by first mapping the reads onto a reference transcriptome (or genome) [46], and then quantifying at the exon, isoform or gene level [47]. If a reference transcriptome is unknown or incomplete, de-novo assembly is an option [48, 49], although the high number of false-positives might make the interpretation unreliable [50, 51]. Count matrices are then normalised using a variety of approaches, making either the assumption that most genes are not differentially expressed [52], or that the same number of genes are up- and down-regulated [53, 54]. Common to most RNA-seq normalisation methods is the correct modelling of the mean-variance relationship, because lowly expressed genes tend to be overdispersed after log-transformation, which would lead to inflated log fold-changes if not modelled properly.

Being a technology that's still maturing, the preprocessing of single-cell RNA-seq is currently a topic of ongoing discussion. Similar initial steps are performed as for RNA-seq, combined with steps that filter out cells based on likelihood of being an actual cell [55], likelihood of being a doublet [56], and/or high expression of stress related genes [57]. There are several approaches to normalise this data. Early methods normalised with cell-specific library sizes and log transformation, similar as with bulk data, which may be calculated using spike-ins [58] or pooled between similar cells [59]. Other approaches will model the expression of a cell as being sampled from a distribution with an overdispersed Poisson and drop-out component. More recently, there has been a shift away from these, driven by the observation that unique molecular identifier (UMI) based data is not zero-inflated [60, 61]. This can create artifacts when log normalising, or overfit when using zero-inflated models. Instead, the newest studies suggest fitting a (regularised) negative binomial or multinomial to better differentiate the technical variation from the biological. Because this is still a very active field, it is hard to tell which normalisation methodology will ultimately gain foothold, given that the very recent benchmarking efforts haven't caught up yet [62]. Nonetheless, it is clear that direct observation of actual data, combined with empirical testing of assumptions, is the way forward.

Differential expression

The most essential question when analysing omics data is: what is different between samples? This is a prototypical question that can be solved by many "classical" statistical tests such as a t-test, analysis of variance or, more general, linear models. Because of the low number of samples in many transcriptomics studies (4 at best, 2 at worst), these tests tend to be very conservative. Moreover, they make assumptions regarding normality of the data, which doesn't hold true for the count nature of RNA-seq data. As discussed before, RNA-seq data has some other peculiarities, such as a strong mean-variance relationship, that either have to be removed at the normalisation step, or incorporated during the testing for differential expression.

Differential expression methods solve these issues by (1) sharing information across similar genes, which allows a better estimate of the variance and in turn makes the statistical testing more powerful and (2) assuming other distributions such as a negative binomial or log-normal distribution. The main difference between the most popular methods such as edgeR and DEseq2 [53, 52, 63] is how these two solutions are implemented in practice. Given the existence of a linear model under the hood, all these methods can handle designs from the simplest pairwise models onto the more complex models with interactions [64].

In single-cell data, differential expression is typically done between two or more cell populations. The most popular tools for single-cell transcriptomics analysis use surprisingly simple tests to answer this question, such as t-tests and Wilcoxon rank-sum tests [65, 66]. Benchmarking has shown that these tests do quite well compared to tools made for bulk RNA-seq or single-cell RNA-seq, with the latter sometimes even performing worse [67]. In the end, what most of these methods output is a ranking of differential expression, but no clear indication of statistical signficance across biological replicates. The latter would require replicated designs and methods able to handle such hierarchical data, a use case still rare in the single-cell omics field [68, 69].

Dimensionality reduction

Omics data typically has more than a thousand features, such as genes, proteins or genomic regions. But because a lot of information is shared between features, for example by co-regulation, the actual number of dimensions is often much lower in the data. As an example, from a biological perspective, a couple of cell type, cell state and cell cycle axes could be enough to understand most of the differences between immune cells in the blood. This is why dimensionality reduction techniques are one of the essential techniques for analysing transcriptomics data [70]. They are used for simplifying the input data for downstream methods, as a way to denoise the data, and for visualisation purposes [71].

The most straightforward way to reduce the dimensions of a dataset is to decompose the data into a product of other matrices, each with lower dimensions then the original. This is the approach followed by linear dimensionality reduction techniques, such as principal component analysis (PCA), independent component analysis (ICA) and some forms of multi-dimensional scaling (MDS). Biological data may also contain non-linear structures, or the number of requested dimensions may be too low to capture the complexity of the data with a linear transformation. This can be handled by non-linear dimensionality reduction techniques such as diffusion map, t-distributed stochastic neighbour embedding (t-SNE), uniform manifold approximation and projection (UMAP) or autoencoders [72].

Broadly speaking, linear methods are often faster and are therefore more useful for denoising and preprocessing. Non-linear methods on the other hand can fit complex manifold structures in the data, and are therefore more useful when a low number of dimensions is required such as for visualisation [71].

Grouping samples, cells and genes

Instead of reducing a dataset to a couple of continuous dimensions, it is also useful to simplify a dataset to a couple of discrete groups of genes, cells or samples. This has a direct biological origin; genes often share function and/or regulators and these modules tend to follow similar expression profiles. Similarly, cells tend to differentiate into functionally similar cell populations with their own characteristic expression profiles. By grouping these genes and cells into modules or populations, the highly dimensional dataset becomes much easier to interpret, and can be functionally analysed.

A clustering method typically first defines a distance or similarity measure, such as the euclidean distance, and then tries to group similar objects such that objects within a group have a relatively low distance [73]. Although clustering is by far the most popular way to group objects, alternatives such as biclustering do exist that do not require objects to be similar in all dimensions [74]. There are countless way to cluster, and the choice of method is primarily driven by the dimensions of the data: clustering of a few thousands of genes across hundred samples requires different tools then clustering millions of cells across these same thousand genes

[75]. Another important factor in transcriptomics is the freedom in similarity metric, given that a correlation is often more appropriate but this is not supported by many classical clustering algorithms.

Co-expression of genes is often, but not always, caused by co-regulation. These common regulators can be found by looking for enrichment of motifs or ChIP peaks in the promoter or enhancer regions associated with the gene. Similar tactics can be employed to find enriched functions within a module by cross-linking it with functional databases such as Gene Ontology and KEGG. This analysis can be used to assign novel functions to genes, ultimately leading to a better understanding of which genes are involved in a particular process [76].

Gene modules can also be used as a feature extraction method for downstream machine learning or statistical analyses. A gene in this regard is often summarised into an average expression value or pseudogene, and this is used to predict a phenotype including patient prognosis or survival [77]. The strength of this approach may lie in its robustness, because the combination of many features may reduce the impact of changes in individual genes [78].

Clusters of biological samples are useful to find new subtypes of patients, and align them with prognostic or phenotypic markers [79]. Conversely, cells can also be clustered to find new subpopulations in a data-driven way [80]. By extracting genes unique to these subpopulations, it then becomes possible to further characterize them. Clustering of cells is also often used as a preprocessing step for other algorithms such as trajectory inference [81] and differential abundance analysis [68].

Trajectory inference

Cells constantly change as a reaction to external or internal signals. This may cause the cell to progress to the next stage of the cell cycle, or make it differentiate into a functional cell type. If the single-cell experiment is designed in such a way that the cells are sampled at random points during this dynamic process, it is possible to reconstruct the dynamics of the transcriptome using trajectory inference (TI) methods. These methods assume that the expression of cells changes gradually, and that it is therefore possible to position cells onto a graph structure such that similar cells are located close to each other. Over 70 methods TI have been published since 2014 (Figure 1.3).

A prototypical TI method combines concepts from both dimensionality reduction and clustering, often by connecting cell clusters based on their closeness in the reduced space [81]. Some methods create a more probabilistic model of differentiation, where each cell is assigned a likelihood of ultimately differentiating into a particular end state [82]. The main differences between TI methods is the type of topology

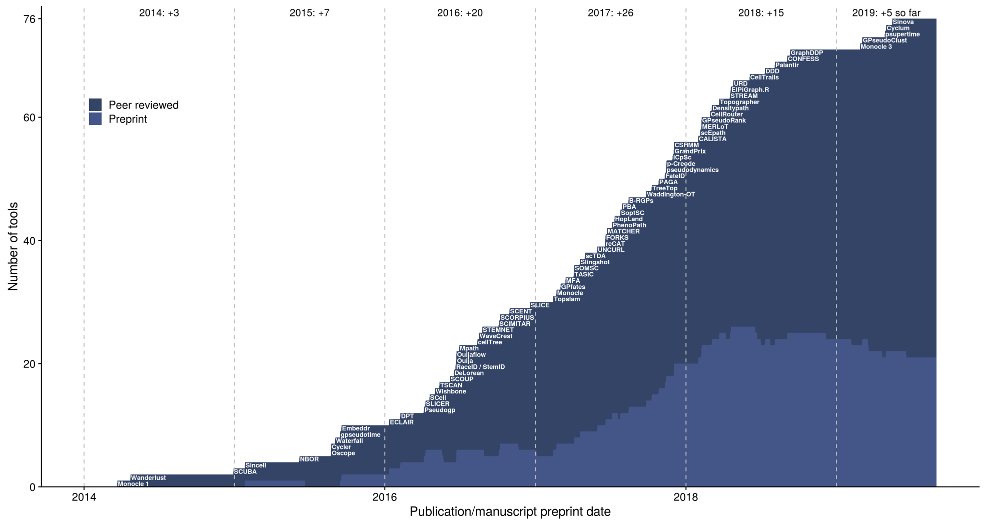


Figure 1.3: Number of trajectory inference tools that have been published over time. Preprint (blue) and peer-reviewed publications (dark blue).

they can detect: early methods mainly focused on linear or circular trajectories, while more recent ones also detect tree or disconnected graphs [81].

TI methods connect cells within a common topology, but it does not provide a direction in which the cells are changing. To get this kind of information, it is necessary to include some other data such as start cells [83], or marker genes that are known to be expressed inside stem-like cells [84]. A recent alternative is to use RNA velocity [85], a technique that can provide for each cell a projection of where the transcriptome is moving in the near future. This technique is based on the idea that single-cell transcriptomics techniques (even those profiling the 3' end), contain both spliced and unspliced reads. For a gene that is increasing in expression, its unspliced reads will be relatively higher than spliced reads, relative to a per-gene ratio that has to be estimated. This makes it possible to create a future expression profile of each cell, and thus to get an idea of the direction that cells are going relative to each other.

Network inference

While trajectory inference gives you relationships between cells, network inference provides you with regulatory relationships between genes. This type of methods will use similarities between expression profiles to find potential regulatory links between pairs of genes. There are many ways to infer a network based on expression profiles, and this often involves the calculation of some similarity between profiles or how predictive the expression of a regulator is for the expression of the target [86].

Expression data on its own is often not informative enough to capture the full complexity of real regulatory networks, especially those in eukaryotes. Two regulators that are co-regulated themselves will have very similar expression patterns, and a network inference algorithm would have a hard time distinguishing their actual target genes. Regulatory networks are so complex and context-specific that using only expression data results in a large number of false positive regulatory links in the data. Often, other information is therefore added to the method, such as presence of transcription factor binding sites close to the promoter [87, 88] or temporal information coming from time series experiments or more recently also RNA velocity [89].

Other methods

There are several other types of methods commonly used in (single-cell) transcriptomics, but that are not used in this thesis and only briefly discussed here.

With complex designs, it often happens that there are unwanted biological or technical effects in the data that cannot be removed through mere normalisation. These can be corrected through the use of batch correction methods [90] or, for single-cell data, cell alignment methods [91, 92]. While they can certainly be useful, batch correction should only be used as a last resort, i.e. when it is impractical to control for these confounders [93]. Cell hashing [94] or cryopreservation may reduce the need for these methods as well.

Batch correction or alignment makes it possible to combine different datasets from the same kind of data but from different biological samples. Related to this are data integration methods, that combine different kinds of data, such as transcriptomics and proteomics, from the same samples [32]. After integration, any previously described method can be applied on the ”joint” dataset [95].

Annotating cell types is typically the first step of interpreting single-cell data, but is now done manually. The more data becomes available, the more it becomes realistic to do this annotation automatically. This can be done in three ways: using bulk data as a reference [96], using single-cell data as a reference [97], or by using marker profiles curated by experts [98]. How new cell types are identified within such an analysis remains challenging, and may in fact turn into a philosophical discussion about what a cell type actually is [32].

On the horizon

The methods that have been described so far can be considered part of the ”basic” toolkit for analysing and modelling transcriptomics data. These methods become even more powerful when combined with each other. Some examples are:

- Combining cell alignment with differential expression to also analyse differential abundance [68]
- Combining TI with data integration to analyse the dependencies between different modalities during cellular dynamics
- Combining network inference with TI to infer dynamic networks

Many more combinations are possible, most of which are still early in development and are not discussed here. Rather, they will be reviewed in a future perspectives chapter (Chapter 6), with a particular focus on TI.

Benchmarking methods in computational biology

The paragraphs above have provided a very broad overview of methods available for transcriptomics research. Once you delve into the specifics of each method, it becomes clear that there’s often debate within a field for what the best approach is to solve a certain problem [1]. For example, most types of single-cell data analyses have dozens of methods available [40] (Figure 1.4).

For an outsider, the availability of so many alternatives can seem daunting. A user will then resort to choosing the method that looks most trustworthy based on proxy metrics such as the journal in which it was published or the popularity amongst its peers. These proxy metrics are however very prone to biases, and might stifle new developments in a very young field because the older tools get more attention [100]. One way to address this is to have an independent comparison (or benchmark) to assess which of the many methods has a high chance of being useful.

Benchmarks in bioinformatics take on many forms. Papers that introduce a method (have to) compare it to previously published methods. This analysis is typically small, showcasing where a method performs better than the current state-of-the-art, and a quantitative comparison is often understandably lacking [101]. In some studies the benchmark is the main story, and contain a comparison of many methods with mostly quantitative metrics.

These metrics encompass

[1]These discussions can sometimes be quite heated as in the kallisto versus Salmon debate [99], further showcasing the need for independent or community-wide comparisons.

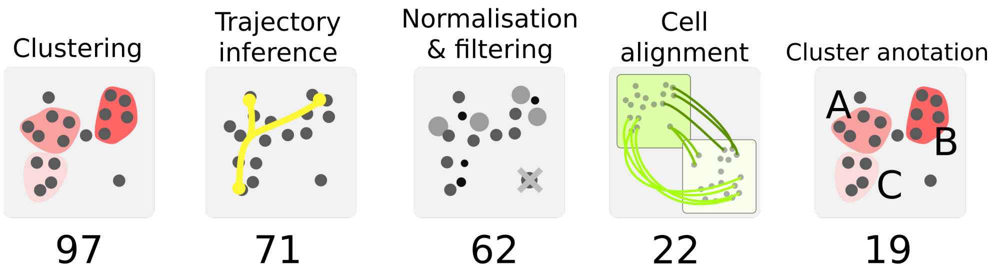


Figure 1.4: Number of methods that are available for a particular kind of single-cell analysis, according to www.scrna-tools.org [40].

1. **Accuracy**. How close is the model to biological reality? This is the most important metric, but also the most difficult to assess. There are many ways to define a gold standard, all of which have imperfections [102]: computational simulations, usage of trusted technologies, or expert manual curation. To compare models, metrics have to be created and validated [103]. Metrics can have biases themselves, and systematic benchmarking therefore assesses multiple metrics [104]. It is often helpful to combine a quantitative assessment with a qualitative look of the results to get a hands-on feeling with how the different metrics work.

2. **Robustness** A robust method produces the same result even if the initial conditions, such as input data and parameters, are slightly altered. This has many facets: changes in the amount of noise, dimensions of input data, stability to approximately the same input data, stability to a different pseudorandom state and robustness to parameters. These questions are often easier to study because each type can be simulated, while making sure models of noise or batch effects match the real life use case.

3. **Scalability** In a fast evolving field such as single-cell genomics, the speed by which the dataset size increases outruns the speed by which our computational resources increase [105]. A benchmarking study therefore often assesses the amount of resources a method uses for a typical dataset, and how it will scale in the future. Running time of a method is most often included, although memory usage and hard drive input/output may also be relevant. An assessment of scalability should ideally be done inside an isolated computer environment, as other running software may strongly affect the result.

4. **Usability** Due to various reasons, the quality of most software in computational biology is poor [106]. Even if such a tool would be very accuracte,

scalable and robust, it would be a nightmare for a user to install and run. An unusable tool may at the basal level result in a minor annoyance for the user, making it hard to apply on their own data. But other usability criteria, such as code availability and unit testing, may have a large impact on the reproducibility and correctness of the bioinformatics results. Assessing the usability is therefore very important for a benchmark, but can be prone to subjectivity. Studies describing best practices often focus on different aspects [107, 108, 109], and the quality of documentation can be in the eye of the beholder. Also, some aspects may be interesting but not essential [110].

5. **Similarity** Just like many blind men are necessary to get a complete picture of how an elephant feels, so we need many methods to get a complete picture of a particular biological system. Some benchmarks therefore assess how different the outputs of methods are, so that users can select methods based on the different kind of view they provide on the same dataset [111, 67].

Goals and problem setting

My main goal in this thesis was to see how I can use transcriptomics to better understand the development and function of immune cells. To answer this question, I investigated three different aspects of transcriptomics analyses: (1) how to characterise the dynamics of immune cells during development, (2) assess which genes are different between progenitors or developmental stages and (3) group these genes into co-expression modules to get a global overview of the main transcriptomic differences between populations.

First, I wanted to understand how cells change during development. TI methods provide one of the best ways to do this at the moment: single-cell RNA-seq datasets have become readily available, and a trajectory model provides an unprecedented overview of the dynamic changes during cell differentiation. However, the sheer number of methods available, together with the challenging high dimensionality of the data, make the application of these methods difficult. It is highly unclear which of the more than 70 methods would provide the most optimal model on a give dataset. This is not only problematic for me, but also for the whole community of users and developers. In chapter 4 of the thesis, I therefore set out to develop a large-scale evaluation of TI methods. This is further complemented with future perspectives chapter 6, in which I try to lay out new methods that may make trajectories easier to interpret in the future.

Secondly, I wanted to investigate which genes are different between differentiation stages and progenitors. Differences between two populations, or through a more complex hierarchical design, are relatively easy to analyse using differential expression methods. However, once there are three or more equally-important populations, the analysis becomes more complicated because of the combinatorial number of necessary pairwise comparisons. In chapter 2, I therefore set out to develop a method that makes it possible to analyse the transcriptomic heterogeneity between three populations. This method is also able to assess the functional heterogeneity of these populations, by looking at which functional gene sets are consistently upregulated in one or two populations.

Finally, I wanted to have a look at the main co-expression modules that drive the differences between immune cell populations. Module detection methods provide the ideal playing ground to do this. However, the many types of methods (clustering, biclustering, decomposition, direct NI and iterative NI) make it difficult for me, other users and developers to rapidly apply the most accurate method on their data. In chapter 3, I set out to develop a comprehensive benchmark of the methods, and use my results to develop guidelines for users and developers. I also used the experience of the two benchmarks within this thesis to highlight my view on the future of benchmarking in computational biology in a future perspectives chapter 7.

References

[1] Yansheng Liu, Andreas Beyer, and Ruedi Aebersold. “On the Dependency of Cellular Protein Levels on mRNA Abundance”. In: *Cell* 165.3 (Apr. 21, 2016), pp. 535–550. ISSN: 1097-4172. DOI: 10.1016/j.cell.2016.03.014.

[2] John T. Lis. “A 50 Year History of Technologies That Drove Discovery in Eukaryotic Transcription Regulation”. In: *Nature Structural & Molecular Biology* 26.9 (Sept. 2019), pp. 777–782. ISSN: 1545-9985. DOI: 10.1038/s41594-019-0288-9.

[3] Moyra Lawrence, Sylvain Daujat, and Robert Schneider. “Lateral Thinking: How Histone Modifications Regulate Gene Expression”. In: *Trends in Genetics* 32.1 (Jan. 1, 2016), pp. 42–56. ISSN: 0168-9525. DOI: 10.1016/j.tig.2015.10.007.

[4] Leah A. Gates, Charles E. Foulds, and Bert W. O’Malley. “Histone Marks in the ’Driver’s Seat’: Functional Roles in Steering the Transcription Cycle”. In: *Trends in Biochemical Sciences* 42.12 (Dec. 2017), pp. 977–989. ISSN: 0968-0004. DOI: 10.1016/j.tibs.2017.10.004.

[5] François Spitz and Eileen E. M. Furlong. “Transcription Factors: From Enhancer Binding to Developmental Control”. In: *Nature Reviews. Genetics* 13.9 (Sept. 2012), pp. 613–626. ISSN: 1471-0064. DOI: 10.1038/nrg3207.

[6] Katherine M. Lelli, Matthew Slattery, and Richard S. Mann. “Disentangling the Many Layers of Eukaryotic Transcriptional Regulation”. In: *Annual Review of Genetics* 46.1 (2012), pp. 43–68. DOI: 10.1146/annurev-genet-110711-155437.

[7] Aimee L. Jalkanen, Stephen J. Coleman, and Jeffrey Wilusz. “Determinants and Implications of mRNA Poly(A) Tail Size - Does This Protein Make My Tail Look Big?” In: *Seminars in cell & developmental biology* 0 (Oct. 2014), pp. 24–32. ISSN: 1084-9521. DOI: 10.1016/j.semcdb.2014.05.018.

[8] Ivano Legnini et al. “FLAM-Seq: Full-Length mRNA Sequencing Reveals Principles of Poly(A) Tail Length Control”. In: *Nature Methods* 16.9 (Sept. 2019), pp. 879–886. ISSN: 1548-7105. DOI: 10.1038/s41592-019-0503-y.

[9] Ross C. Wilson and Jennifer A. Doudna. “Molecular Mechanisms of RNA Interference”. In: *Annual Review of Biophysics* 42.1 (2013), pp. 217–239. DOI: 10.1146/annurev-biophys-083012-130404.

[10] Georgi K. Marinov et al. “From Single-Cell to Cell-Pool Transcriptomes: Stochasticity in Gene Expression and RNA Splicing”. In: *Genome Research* 24.3 (Jan. 3, 2014), pp. 496–510. ISSN: 1088-9051, 1549-5469. DOI: 10.1101/gr.161034.113.

[11] Valentine Svensson et al. “Power Analysis of Single-Cell RNA-Sequencing Experiments”. In: *Nature Methods* 14.4 (Apr. 2017), pp. 381–387. ISSN: 1548-7105. DOI: 10.1038/nmeth.4220.

[12] *Comparative Performance of the BGI and Illumina Sequencing Technology for Single-Cell RNA-Sequencing | bioRxiv*. URL: https://www.biorxiv.org/content/10.1101/552588v2 (visited on 05/27/2019).

[13] Zichen Wang, Alexander Lachmann, and Avi Ma’ayan. “Mining Data and Metadata from the Gene Expression Omnibus”. In: *Biophysical Reviews* 11.1 (Feb. 1, 2019), pp. 103–110. ISSN: 1867-2469. DOI: 10.1007/s12551-018-0490-8.

[14] Marc Sultan et al. “Influence of RNA Extraction Methods and Library Selection Schemes on RNA-Seq Data”. In: *BMC genomics* 15 (Aug. 11, 2014), p. 675. ISSN: 1471-2164. DOI: 10.1186/1471-2164-15-675.

[15] Hélène Lopez-Maestre et al. “SNP Calling from RNA-Seq Data without a Reference Genome: Identification, Quantification, Differential Analysis and Impact on the Protein Sequence”. In: *Nucleic Acids Research* 44.19 (Nov. 2, 2016), e148. ISSN: 0305-1048. DOI: 10.1093/nar/gkw655.

[16] Marco De Simone, Grazisa Rossetti, and Massimiliano Pagani. “Single Cell T Cell Receptor Sequencing: Techniques and Future Challenges”. In: *Frontiers in Immunology* 9 (July 18, 2018). ISSN: 1664-3224. DOI: 10.3389/fimmu.2018.01638.

[17] Stavros Bashiardes, Gili Zilberman-Schapira, and Eran Elinav. “Use of Metatranscriptomics in Microbiome Research”. In: *Bioinformatics and Biology Insights* 10 (Apr. 20, 2016), pp. 19–25. ISSN: 1177-9322. DOI: 10.4137/BBI.S34610.

[18] Anne Senabouth et al. “Comparative Performance of the BGI and Illumina Sequencing Technology for Single-Cell RNA-Sequencing”. In: *bioRxiv* (Feb. 24, 2019), p. 552588. DOI: 10.1101/552588.

[19] Tuomo Mantere, Simone Kersten, and Alexander Hoischen. “Long-Read Sequencing Emerging in Medical Genetics”. In: *Frontiers in Genetics* 10 (2019). ISSN: 1664-8021. DOI: 10.3389/fgene.2019.00426.

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

[21] Philip Brennecke et al. “Accounting for Technical Noise in Single-Cell RNA-Seq Experiments”. In: *Nature Methods* 10.11 (Nov. 2013), pp. 1093–1095. ISSN: 1548-7105. DOI: 10.1038/nmeth.2645.

[22] Simone Picelli et al. “Smart-Seq2 for Sensitive Full-Length Transcriptome Profiling in Single Cells”. In: *Nature Methods* 10.11 (Nov. 2013), pp. 1096–1098. ISSN: 1548-7105. DOI: 10.1038/nmeth.2639.

[23] Sanja Vickovic et al. “Massive and Parallel Expression Profiling Using Microarrayed Single-Cell Sequencing”. In: *Nature Communications* 7 (Oct. 14, 2016), p. 13182. ISSN: 2041-1723. DOI: 10.1038/ncomms13182.

[24] Evan Z. Macosko et al. “Highly Parallel Genome-Wide Expression Profiling of Individual Cells Using Nanoliter Droplets”. In: *Cell* 161.5 (May 21, 2015), pp. 1202–1214. ISSN: 1097-4172. DOI: 10.1016/j.cell.2015.05.002.

[25] Allon M. Klein et al. “Droplet Barcoding for Single-Cell Transcriptomics Applied to Embryonic Stem Cells”. In: *Cell* 161.5 (May 21, 2015), pp. 1187–1201. ISSN: 1097-4172. DOI: 10.1016/j.cell.2015.04.044.

[26] Sayantan Bose et al. “Scalable Microfluidics for Single-Cell RNA Printing and Sequencing”. In: *Genome Biology* 16.1 (June 6, 2015), p. 120. ISSN: 1465-6906. DOI: 10.1186/s13059-015-0684-3.

[27] Todd M. Gierahn et al. “Seq-Well: Portable, Low-Cost RNA Sequencing of Single Cells at High Throughput”. In: *Nature Methods* 14.4 (Apr. 2017), pp. 395–398. ISSN: 1548-7105. DOI: 10.1038/nmeth.4179.

[28] Alexander B. Rosenberg et al. “Single-Cell Profiling of the Developing Mouse Brain and Spinal Cord with Split-Pool Barcoding”. In: *Science* 360.6385 (Apr. 13, 2018), pp. 176–182. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.aam8999.

[29] Samuel G. Rodriques et al. “Slide-Seq: A Scalable Technology for Measuring Genome-Wide Expression at High Spatial Resolution”. In: *Science* 363.6434 (Mar. 29, 2019), pp. 1463–1467. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.aaw1219.

[30] Grace X. Y. Zheng et al. “Massively Parallel Digital Transcriptional Profiling of Single Cells”. In: *Nature Communications* 8 (Jan. 16, 2017), p. 14049. ISSN: 2041-1723. DOI: 10.1038/ncomms14049.

[31] Diego Adhemar Jaitin et al. “Massively Parallel Single Cell RNA-Seq for Marker-Free Decomposition of Tissues into Cell Types”. In: *Science (New York, N.Y.)* 343.6172 (Feb. 14, 2014), pp. 776–779. ISSN: 0036-8075. DOI: 10.1126/science.1247651.

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

[33] Sarah A. Vitak et al. “Sequencing Thousands of Single-Cell Genomes with Combinatorial Indexing”. In: *Nature Methods* 14.3 (Mar. 2017), pp. 302–308. ISSN: 1548-7105. DOI: 10.1038/nmeth.4154.

[34] Blue B. Lake et al. “Integrative Single-Cell Analysis of Transcriptional and Epigenetic States in the Human Adult Brain”. In: *Nature Biotechnology* 36.1 (Jan. 2018), pp. 70–80. ISSN: 1546-1696. DOI: 10.1038/nbt.4038.

[35] Harrison Specht et al. “High-Throughput Single-Cell Proteomics Quantifies the Emergence of Macrophage Heterogeneity”. In: *bioRxiv* (June 9, 2019), p. 665307. DOI: 10.1101/665307.

[36] Marlon Stoeckius et al. “Simultaneous Epitope and Transcriptome Measurement in Single Cells”. In: *Nature Methods* 14.9 (Sept. 2017), pp. 865–868. ISSN: 1548-7105. DOI: 10.1038/nmeth.4380.

[37] A. Sina Booeshaghi et al. “Design Principles for Open Source Bioinstrumentation: The Poseidon Syringe Pump System as an Example”. In: *bioRxiv* (Jan. 17, 2019), p. 521096. DOI: 10.1101/521096.

[38] Arnav Moudgil. *Multimodal scRNA-Seq.* Feb. 25, 2019. DOI: 10.5281/zenodo.2628012.

[39] Eleni P. Mimitou et al. “Multiplexed Detection of Proteins, Transcriptomes, Clonotypes and CRISPR Perturbations in Single Cells”. In: *Nature Methods* 16.5 (May 2019), p. 409. ISSN: 1548-7105. DOI: 10.1038/s41592-019-0392-0.

[40] Luke Zappia, Belinda Phipson, and Alicia Oshlack. “Exploring the Single-Cell RNA-Seq Analysis Landscape with the scRNA-Tools Database”. In: *bioRxiv* (Oct. 20, 2017), p. 206573. DOI: 10.1101/206573.

[41] B. M. Bolstad et al. “A Comparison of Normalization Methods for High Density Oligonucleotide Array Data Based on Variance and Bias”. In: *Bioinformatics (Oxford, England)* 19.2 (Jan. 22, 2003), pp. 185–193. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/19.2.185.

[42] Zhijin Wu. “A Review of Statistical Methods for Preprocessing Oligonucleotide Microarrays”. In: *Statistical methods in medical research* 18.6 (Dec. 2009), pp. 533–541. ISSN: 0962-2802. DOI: 10.1177/0962280209351924.

[43] Rob Patro et al. “Salmon: Fast and Bias-Aware Quantification of Transcript Expression Using Dual-Phase Inference”. In: *Nature methods* 14.4 (Apr. 2017), pp. 417–419. ISSN: 1548-7091. DOI: 10.1038/nmeth.4197.

[44] Nicolas L. Bray et al. “Near-Optimal Probabilistic RNA-Seq Quantification”. In: *Nature Biotechnology* 34.5 (May 2016), pp. 525–527. ISSN: 1546-1696. DOI: 10.1038/nbt.3519.

[45] Douglas C. Wu et al. “Limitations of Alignment-Free Tools in Total RNA-Seq Quantification”. In: *BMC Genomics* 19.1 (July 3, 2018), p. 510. ISSN: 1471-2164. DOI: 10.1186/s12864-018-4869-5.

[46] Alexander Dobin et al. “STAR: Ultrafast Universal RNA-Seq Aligner”. In: *Bioinformatics* 29.1 (Jan. 2013), pp. 15–21. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bts635.

[47] Simon Anders, Paul Theodor Pyl, and Wolfgang Huber. “HTSeq—a Python Framework to Work with High-Throughput Sequencing Data”. In: *Bioinformatics* 31.2 (Jan. 15, 2015), pp. 166–169. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btu638.

[48] Manfred G. Grabherr et al. “Full-Length Transcriptome Assembly from RNA-Seq Data without a Reference Genome”. In: *Nature Biotechnology* 29.7 (May 15, 2011), pp. 644–652. ISSN: 1546-1696. DOI: 10.1038/nbt.1883.

[49] Cole Trapnell et al. “Differential Gene and Transcript Expression Analysis of RNA-Seq Experiments with TopHat and Cufflinks”. In: *Nature Protocols* 7.3 (), p. 562. DOI: 10.1038/nprot.2012.016.

[50] Seqc/Maqc-Iii Consortium et al. “A Comprehensive Assessment of RNA-Seq Accuracy, Reproducibility and Information Content by the Sequencing Quality Control Consortium”. In: *Nature Biotechnology* 32.9 (Sept. 2014), pp. 903–914. ISSN: 1546-1696. DOI: 10.1038/nbt.2957.

[51] Adam H. Freedman, Michele Clamp, and Timothy B. Sackton. “Error, Noise and Bias in de Novo Transcriptome Assemblies”. In: *bioRxiv* (Apr. 30, 2019), p. 585745. DOI: 10.1101/585745.

[52] Michael I Love, Wolfgang Huber, and Simon Anders. “Moderated Estimation of Fold Change and Dispersion for RNA-Seq Data with DESeq2”. In: *Genome Biology* 15.12 (2014). ISSN: 1465-6906. DOI: 10.1186/s13059-014-0550-8.

[53] Charity W. Law et al. “Voom: Precision Weights Unlock Linear Model Analysis Tools for RNA-Seq Read Counts”. In: *Genome Biology* 15.2 (Feb. 3, 2014), R29. ISSN: 1474-760X. DOI: 10.1186/gb-2014-15-2-r29.

[54] Marie-Agnès Dillies et al. “A Comprehensive Evaluation of Normalization Methods for Illumina High-Throughput RNA Sequencing Data Analysis”. In: *Briefings in Bioinformatics* 14.6 (Nov. 2013), pp. 671–683. ISSN: 1477-4054. DOI: 10.1093/bib/bbs046.

[55] Aaron T. L. Lun et al. “EmptyDrops: Distinguishing Cells from Empty Droplets in Droplet-Based Single-Cell RNA Sequencing Data”. In: *Genome Biology* 20.1 (Mar. 22, 2019), p. 63. ISSN: 1474-760X. DOI: 10.1186/s13059-019-1662-y.

[56] Christopher S. McGinnis, Lyndsay M. Murrow, and Zev J. Gartner. “DoubletFinder: Doublet Detection in Single-Cell RNA Sequencing Data Using Artificial Nearest Neighbors”. In: *Cell Systems* 8.4 (Apr. 24, 2019), 329–337.e4. ISSN: 2405-4712. DOI: 10.1016/j.cels.2019.03.003.

[57] 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. 31, 2016), p. 2122. ISSN: 2046-1402. DOI: 10.12688/f1000research.9501.2.

[58] Catalina A. Vallejos, John C. Marioni, and Sylvia Richardson. “BASiCS: Bayesian Analysis of Single-Cell Sequencing Data”. In: *PLOS Computational Biology* 11.6 (June 24, 2015), e1004333. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1004333.

[59] Aaron T. L. Lun, Karsten Bach, and John C. Marioni. “Pooling across Cells to Normalize Single-Cell RNA Sequencing Data with Many Zero Counts”. In: *Genome Biology* 17.1 (Apr. 27, 2016), p. 75. ISSN: 1474-760X. DOI: 10.1186/s13059-016-0947-7.

[60] Justin D. Silverman et al. “Naught All Zeros in Sequence Count Data Are the Same”. In: *bioRxiv* (Nov. 26, 2018), p. 477794. DOI: 10.1101/477794.

[61] Valentine Svensson. “Droplet scRNA-Seq Is Not Zero-Inflated”. In: *bioRxiv* (Mar. 19, 2019), p. 582064. DOI: 10.1101/582064.

[62] Luyi Tian et al. “Benchmarking Single Cell RNA-Sequencing Analysis Pipelines Using Mixture Control Experiments”. In: *Nature Methods* 16.6 (June 2019), p. 479. ISSN: 1548-7105. DOI: 10.1038/s41592-019-0425-8.

[63] Mark D. Robinson, Davis J. McCarthy, and Gordon K. Smyth. “edgeR: A Bioconductor Package for Differential Expression Analysis of Digital Gene Expression Data”. In: *Bioinformatics* 26.1 (Jan. 1, 2010), pp. 139–140. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btp616.

[64] Nicholas J. Schurch et al. “How Many Biological Replicates Are Needed in an RNA-Seq Experiment and Which Differential Expression Tool Should You Use?” In: *RNA* 22.6 (Jan. 6, 2016), pp. 839–851. ISSN: 1355-8382, 1469-9001. DOI: 10.1261/rna.053959.115.

[65] Andrew Butler et al. “Integrating Single-Cell Transcriptomic Data across Different Conditions, Technologies, and Species”. In: *Nature Biotechnology* 36.5 (May 2018), pp. 411–420. ISSN: 1546-1696. DOI: 10.1038/nbt.4096.

[66] F. Alexander Wolf, Philipp Angerer, and Fabian J. Theis. “SCANPY: Large-Scale Single-Cell Gene Expression Data Analysis”. In: *Genome Biology* 19.1 (Feb. 6, 2018), p. 15. ISSN: 1474-760X. DOI: 10.1186/s13059-017-1382-0.

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

[68] Lukas M. Weber et al. “Diffcyt: Differential Discovery in High-Dimensional Cytometry via High-Resolution Clustering”. In: *Communications Biology* 2.1 (May 14, 2019), p. 183. ISSN: 2399-3642. DOI: 10.1038/s42003-019-0415-5.

[69] Chamith Y. Fonseka et al. “Mixed-Effects Association of Single Cells Identifies an Expanded Effector CD4+ T Cell Subset in Rheumatoid Arthritis”. In: *Science Translational Medicine* 10.463 (Oct. 17, 2018), eaaq0305. ISSN: 1946-6234, 1946-6242. DOI: 10.1126/scitranslmed.aaq0305.

[70] Allon Wagner, Aviv Regev, and Nir Yosef. “Revealing the Vectors of Cellular Identity with Single-Cell Genomics”. In: *Nature biotechnology* 34.11 (Nov. 8, 2016), pp. 1145–1160. ISSN: 1087-0156. DOI: 10.1038/nbt.3711.

[71] Shiquan Sun et al. “Accuracy, Robustness and Scalability of Dimensionality Reduction Methods for Single Cell RNAseq Analysis”. In: *bioRxiv* (May 17, 2019), p. 641142. DOI: 10.1101/641142.

[72] Gökcen Eraslan et al. “Single-Cell RNA-Seq Denoising Using a Deep Count Autoencoder”. In: *Nature Communications* 10.1 (Jan. 23, 2019), p. 390. ISSN: 2041-1723. DOI: 10.1038/s41467-018-07931-2.

[73] Jeremy J Jay et al. “A Systematic Comparison of Genome-Scale Clustering Algorithms”. In: *BMC Bioinformatics* 13 (Suppl 10 June 25, 2012), S7. ISSN: 1471-2105. DOI: 10.1186/1471-2105-13-S10-S7.

[74] Amela Prelić et al. “A Systematic Comparison and Evaluation of Biclustering Methods for Gene Expression Data”. In: *Bioinformatics (Oxford, England)* 22.9 (May 1, 2006), pp. 1122–1129. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btl060.

[75] Christian Wiwie, Jan Baumbach, and Richard Röttger. “Comparing the Performance of Biomedical Clustering Methods”. In: *Nature Methods* 12.11 (Nov. 2015), pp. 1033–1038. ISSN: 1548-7105. DOI: 10.1038/nmeth.3583.

[76] Damien Chaussabel and Nicole Baldwin. “Democratizing Systems Immunology with Modular Transcriptional Repertoire Analyses”. In: *Nature Reviews. Immunology* 14.4 (Apr. 2014), pp. 271–280. ISSN: 1474-1741. DOI: 10.1038/nri3642.

[77] Peter Langfelder and Steve Horvath. “WGCNA: An R Package for Weighted Correlation Network Analysis”. In: *BMC bioinformatics* 9 (Dec. 29, 2008), p. 559. ISSN: 1471-2105. DOI: 10.1186/1471-2105-9-559.

[78] Juan Xie et al. “It Is Time to Apply Biclustering: A Comprehensive Review of Biclustering Applications in Biological and Biomedical Data”. In: *Briefings in Bioinformatics* (). DOI: 10.1093/bib/bby014.

[79] The Cancer Genome Atlas Network. “Comprehensive Genomic Characterization of Head and Neck Squamous Cell Carcinomas”. In: *Nature* 517.7536 (Jan. 2015), pp. 576–582. ISSN: 1476-4687. DOI: 10.1038/nature14129.

[80] Vladimir Yu Kiselev et al. “SC3: Consensus Clustering of Single-Cell RNA-Seq Data”. In: *Nature Methods* 14.5 (May 2017), pp. 483–486. ISSN: 1548-7105. DOI: 10.1038/nmeth.4236.

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

[82] 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. 3, 2017), eaal2192. ISSN: 2470-9468. DOI: 10.1126/sciimmunol.aal2192.

[83] Sean C. Bendall et al. “Single-Cell Trajectory Detection Uncovers Progression and Regulatory Coordination in Human B Cell Development”. In: *Cell* 157.3 (Apr. 24, 2014), pp. 714–725. ISSN: 0092-8674. DOI: 10.1016/j.cell.2014.04.005.

[84] 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.

[85] 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.

[86] Daniel Marbach et al. “Wisdom of Crowds for Robust Gene Network Inference”. In: *Nature Methods* 9.8 (July 15, 2012), pp. 796–804. ISSN: 1548-7105. DOI: 10.1038/nmeth.2016.

[87] Daniel Marbach et al. “Tissue-Specific Regulatory Circuits Reveal Variable Modular Perturbations across Complex Diseases”. In: *Nature Methods* 13.4 (Apr. 2016), pp. 366–370. ISSN: 1548-7105. DOI: 10.1038/nmeth.3799.

[88] Sara Aibar et al. “SCENIC: Single-Cell Regulatory Network Inference and Clustering”. In: *Nature Methods* 14.11 (Nov. 2017), pp. 1083–1086. ISSN: 1548-7105. DOI: 10.1038/nmeth.4463.

[89] Xiaojie Qiu et al. “Towards Inferring Causal Gene Regulatory Networks from Single Cell Expression Measurements”. In: *bioRxiv* (Sept. 25, 2018), p. 426981. DOI: 10.1101/426981.

[90] W. Evan Johnson, Cheng Li, and Ariel Rabinovic. “Adjusting Batch Effects in Microarray Expression Data Using Empirical Bayes Methods”. In: *Biostatistics (Oxford, England)* 8.1 (Jan. 2007), pp. 118–127. ISSN: 1465-4644. DOI: 10.1093/biostatistics/kxj037.

[91] Ilya Korsunsky et al. “Fast, Sensitive, and Accurate Integration of Single Cell Data with Harmony”. In: *bioRxiv* (Nov. 5, 2018), p. 461954. DOI: 10.1101/461954.

[92] Tim Stuart et al. “Comprehensive Integration of Single-Cell Data”. In: *Cell* 0.0 (June 6, 2019). ISSN: 0092-8674, 1097-4172. DOI: 10.1016/j.cell.2019.05.031.

[93] Vegard Nygaard, Einar Andreas Rødland, and Eivind Hovig. “Methods That Remove Batch Effects While Retaining Group Differences May Lead to Exaggerated Confidence in Downstream Analyses”. In: *Biostatistics (Oxford, England)* 17.1 (Jan. 2016), pp. 29–39. ISSN: 1465-4644. DOI: 10.1093/biostatistics/kxv027.

[94] Marlon Stoeckius et al. “Cell Hashing with Barcoded Antibodies Enables Multiplexing and Doublet Detection for Single Cell Genomics”. In: *Genome Biology* 19.1 (Dec. 19, 2018), p. 224. ISSN: 1474-760X. DOI: 10.1186/s13059-018-1603-1.

[95] Ricard Argelaguet et al. “Multi-Omics Factor Analysis—a Framework for Unsupervised Integration of Multi-omics Data Sets”. In: *Molecular Systems Biology* 14.6 (June 1, 2018), e8124. ISSN: 1744-4292, 1744-4292. DOI: 10.15252/msb.20178124.

[96] Dvir Aran et al. “Reference-Based Analysis of Lung Single-Cell Sequencing Reveals a Transitional Profibrotic Macrophage”. In: *Nature Immunology* 20.2 (Feb. 2019), p. 163. ISSN: 1529-2916. DOI: 10.1038/s41590-018-0276-y.

[97] Amit Frishberg et al. “Cell Composition Analysis of Bulk Genomics Using Single-Cell Data”. In: *Nature Methods* 16.4 (Apr. 2019), p. 327. ISSN: 1548-7105. DOI: 10.1038/s41592-019-0355-5.

[98] Allen W. Zhang et al. “Probabilistic Cell Type Assignment of Single-Cell Transcriptomic Data Reveals Spatiotemporal Microenvironment Dynamics in Human Cancers”. In: *bioRxiv* (Jan. 16, 2019), p. 521914. DOI: 10.1101/521914.

[99] Titus Brown. *Thoughts on My Salmon Review*. URL: http://ivory.idyll.org/blog/2017-thinking-back-on-salmon-reviwe.html (visited on 05/29/2019).

[100] Paul P. Gardner et al. “A Meta-Analysis of Bioinformatics Software Benchmarks Reveals That Publication-Bias Unduly Influences Software Accuracy”. In: *bioRxiv* (Jan. 2, 2017), p. 092205. DOI: 10.1101/092205.

[101] Lukas M. Weber et al. “Essential Guidelines for Computational Method Benchmarking”. In: (Dec. 3, 2018).

[102] Serghei Mangul et al. “Systematic Benchmarking of Omics Computational Tools”. In: *Nature Communications* 10.1 (Mar. 27, 2019), p. 1393. ISSN: 2041-1723. DOI: 10.1038/s41467-019-09406-4.

[103] Curtis Huttenhower et al. “The Impact of Incomplete Knowledge on Evaluation: An Experimental Benchmark for Protein Function Prediction”. In: *Bioinformatics (Oxford, England)* 25.18 (Sept. 15, 2009), pp. 2404–2410. ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btp397.

[104] 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 (Jan. 1, 2011), p. 537. ISSN: 1744-4292, 1744-4292. DOI: 10.1038/msb.2011.70.

[105] 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.

[106] Serghei Mangul et al. “A Comprehensive Analysis of the Usability and Archival Stability of Omics Computational Tools and Resources”. In: *bioRxiv* (Oct. 25, 2018), p. 452532. DOI: 10.1101/452532.

[107] Greg Wilson et al. “Best Practices for Scientific Computing”. In: *PLOS Biology* 12.1 (Jan. 7, 2014), e1001745. ISSN: 1545-7885. DOI: 10.1371/journal.pbio.1001745.

[108] Morgan Taschuk and Greg Wilson. “Ten Simple Rules for Making Research Software More Robust”. In: *PLOS Computational Biology* 13.4 (Apr. 13, 2017), e1005412. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1005412.

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

[110] Greg Wilson et al. “Good Enough Practices in Scientific Computing”. In: *PLOS Computational Biology* 13.6 (June 22, 2017), e1005510. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1005510.

[111] Daniel Marbach et al. “Wisdom of Crowds for Robust Gene Network Inference”. In: *Nature methods* 9.8 (July 15, 2012), pp. 796–804. ISSN: 1548-7091. DOI: 10.1038/nmeth.2016.

2 | Analysing the transcriptome of three biological conditions

Contributions
The experimental work discussed in this chapter was conducted by Lianne van de Laar and collaborators, as specified in the research paper [1]. The initial computational analysis of the gene expression data was conducted by Liesbet Martens. The development of the triwise methods together with its application was developed by Wouter Saelens, in a close feedback loop with Liesbet Martens, Lianne van de Laar and Martin Guilliams.

Introduction

Transcriptome profiling can be a powerful tool to functionally characterise different cell populations. On the one hand, differential expression coupled with functional enrichment can be used to find the cellular functions changing when cells differentiate or react to external stimuli. On the other hand, co-expression modules can be used to create an overview of how functions change among dozens of cellular conditions. Both approaches are now established workflows to decipher functional diversity between two or many cell populations.

However, in some cases, researchers are only interested in comparing three conditions. For example, one can have two main conditions of interest and a third reference condition. Based on pairwise differential expression alone it can be hard to interpret what functional changes are common to both cells compared to the reference, or whether some are more specifically up- or downregulated. Other scenarios include cell differentiation, where one progenitor can give rise to two distinct daughter cell populations, or two chemical/genetic perturbations with one common initial condition. Some examples of this that we encountered within the department are provided in Figure 2.1.

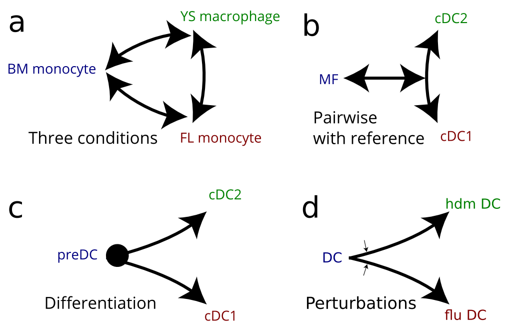


Figure 2.1: Different experimental setups where triwise can be useful. **a** When comparing three related conditions, such as the three main progenitors of macrophages in mice: bone marrow monocytes (BM monocyte), yolk-sac macrophages (YS macrophage) and fetal-liver monocytes (FL monocyte) **b** When one wants to compare the difference between two cell populations in relation to a third related population: macrophages (MF) compared to two types of conventional dendritic cells (cDC). **c** When studying the expression of two progenitor populations in relation to its precursor: precursor dendritic cells (pre-DC) compared to the two types of conventional dendritic cells (cDC). **d** When comparing two perturbations in relation to the expression before the perturbation: dendritic cells (DC) before and after treatment with house-dust mite (hdm) or flu.

To be able to handle these experimental designs, we developed triwise, a visualisation and analysis workflow to simultaneously assess the functional diversity of three biological conditions.

Triwise methodology

Let's say we want to compare two conditions. Often, we're not interested in how high the expression is in either condition, but more in how much the expression changes. This is measured as the fold-change between the two average expression values. One way to visualise this value is to plot the log expression values as a two-dimensional scatterplot (Figure 2.2a), and project each gene onto the line formed by $x + y = 0$ (or in its alternative form, $y = -x$). The distance from the origin then represents the log-fold change of this gene, and any information about the absolute expression of a gene is removed. If we rotate this projection +45° such that it becomes identical to the x-axis, we essentially reduced our dimensions from two to one.

It is possible to do a similar projection and rotation for any number of conditions, by projecting on the hyperplane formed by the equation $x_1 + x_2 + ... + x_n = 0$. This is

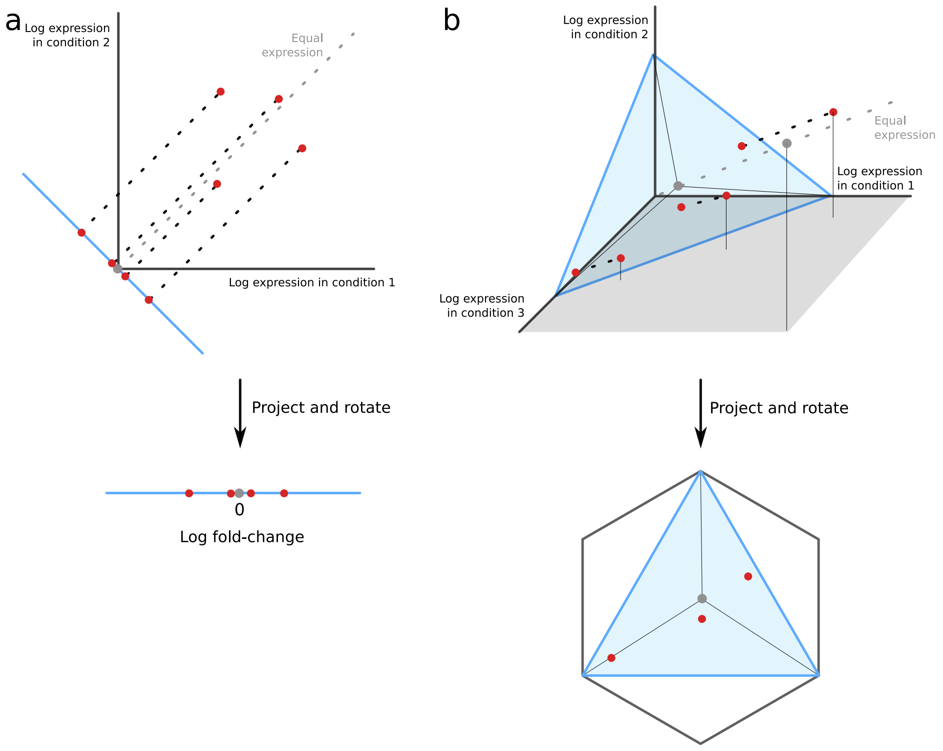

a
Log expression in condition 2
Equal expression
Log expression in condition 1
Project and rotate
0
Log fold-change
b
Log expression in condition 2
Equal expression
Log expression in condition 1
Log expression in condition 3
Project and rotate

Figure 2.2: Illustration of the barycentric transformation of expression data in two and three dimensions.

particularly interesting in three dimensions, because this would make it possible to visualise the expression changes between three conditions in two dimensions. This projection and rotation is also known as a transformation to a barycentric coordinate system [1]. This transformation can be done on any plane, but here we chose to do the transformation such that points $(1, 0, 0)$, $(0, 1, 0)$ and $(0, 0, 1)$ are projected onto a equilateral triangle with base the x-axis (Figure 2.2b):

$$B = X \times \begin{bmatrix} 1 & 0 \\ \cos \frac{2}{3}\pi & \sin \frac{2}{3}\pi \\ \cos \frac{-2}{3}\pi & \sin \frac{-2}{3}\pi \end{bmatrix}$$

For ease of interpretability, we add hexagon gridlines to the visualisation, where points lying on such a hexagon have the same fold-change between their highest and lowest expression values among the three conditions. The transformed expression values can then be plotted in a 2D space on which annotations can provide information about individual genes (Figure 2.3a) or functional gene sets (Figure 2.3b,c).

To understand how the three conditions differ in a functional way, it would be useful if we could assess which of a set of predefined functional gene sets was upregulated in one or more conditions. This is an analysis that is already often done, with both gene modules (hypergeometric test, fisher's exact test) and pairwise comparisons (numerous tests including gene set enrichment analysis [2]). We developed two such tests to be used together with barycentric coordinates, by using elements from directional statistics [3]. Both tests will first calculate a mean vector of a given gene set (with a particular direction and length), and assess the probability of observing this mean vector given a particular null model.

The self-contained test assesses the hypothesis that the direction of differentially expressed genes within the gene set originated from a uniform distribution or whether there is a significant trend towards the mean angle. For this purpose we use the Rayleigh z-test, which is a test that has been previously used to analyse the directional uniformity of birds or bees. The test-statistics of the Rayleigh z-test is calculated by first normalising the vectors of each gene to the unit vector: $\vec{b_u} = \frac{\vec{b}}{||\vec{b}||}$, and then taking the mean resultant length of these vectors across genes: $R = \frac{||\sum_i \vec{b_u}_i||}{n}$. The p-values of a Rayleigh z-test can be rapidly approximated using Rayleigh's approximation [4]:

[1]Probably the most well known use case of the barycentric coordinate systems are ternary plots, used for many applications to represent fractions between three variables such as soil composition (Clay, Sand and Silt).

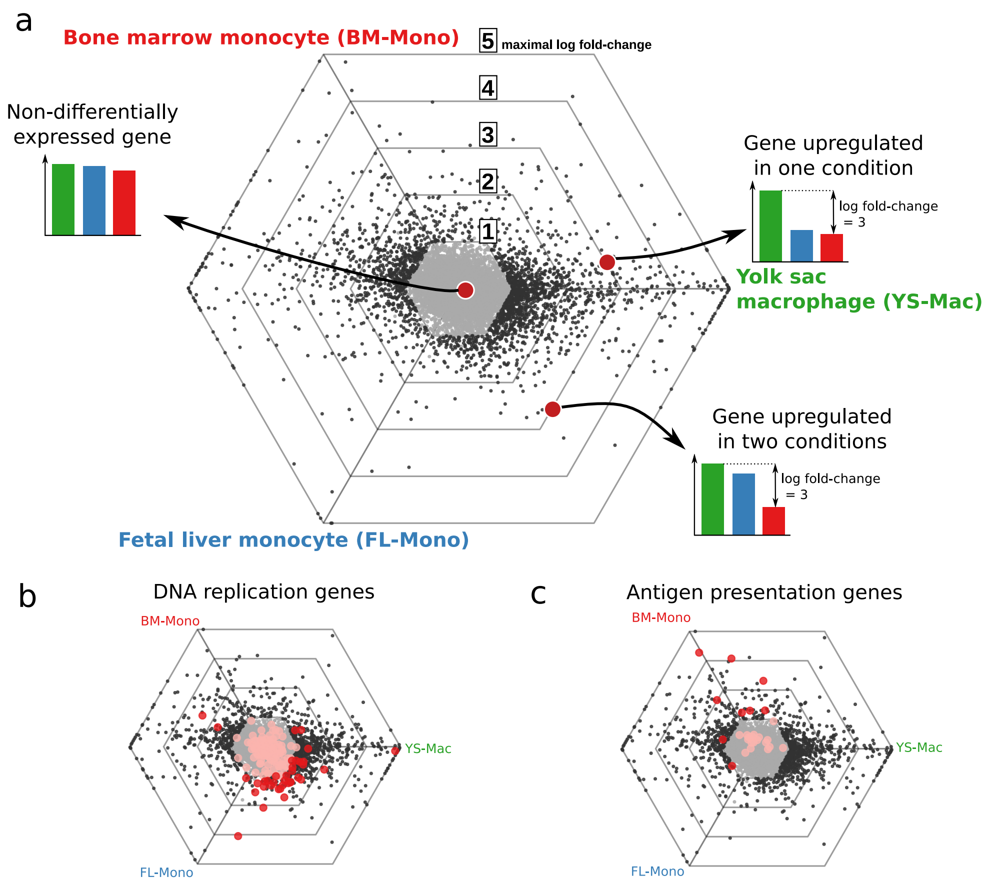


Figure 2.3: Application of triwise on expression data from three macrophage progenitor cells (Fetal liver monocytes, bone marrow monocytes and yolk-sac macrophages). a: Triwise visualises the expression differences between three conditions. Points lying on a hexagonal grid have the same maximal fold-change between any two conditions. **(b-c)**: Gene sets can be unidirectionally upregulated in one or two conditions.

$$p = \begin{cases} e^{-Z}\left[1+\frac{1}{2}+\frac{2Z-Z^2}{4n}-\frac{24Z-132Z^2+76Z^3-9Z^4}{288n^2}\right], & \text{if } n < 50 \\ e^{-Z}, & \text{otherwise} \end{cases}$$

This test does not take into account the strength of upregulation, nor does it consider the underlying preference of all genes towards a particular direction. We therefore also developed a competitive gene-set test, where we use a gene resampling procedure to estimate the significance of an upregulation of a gene set towards its mean direction. To calculate the strength and direction of upregulation, we calculate the vector sum of the gene positions within the barycentric coordinate system, and convert it to a polar coordinate with length z and average direction θ.

To make the statistical test more robust to outliers (because of genes with high log fold-changes), it is possible to first transform the magnitude of the barycentric vectors to more robust measures such as the rank of the norm or an indicator function for differential expression of a gene.

We compare this test statistic with those obtained from random gene samples of the same size as the gene set of interest. To let the null-distribution depend on the direction of upregulation, we weigh each sample using a von Mises distribution [4]. This allows random samples closer to the mean angle of a gene set to have a higher influence on the resulting p-value. Let z and θ be the test statistic and mean angle from the gene set of interest and z_i and θ_i those from a random sample:

$$Pr(Z \geq \ z|H_0, \theta) = \frac{\sum_{i|z \geq z_i} w(\theta_i)}{\sum_i w(\theta_i)}$$

$$w(\theta_i|\theta, \kappa) \propto e^{\kappa \cos{(\theta - \theta_i)}}$$

where z_i and θ_i are from different independent random samples of genes with the same size as the gene set of interest. A high number of samples will lead to more accurate estimations of the p-value, at the cost of higher computing time. κ is an additional parameter denoting the concentration of the von Mises distribution. Higher values of κ will make the null distribution more sensitive to local differences in the distribution of the genes. This is illustrated in Figure 2.4.

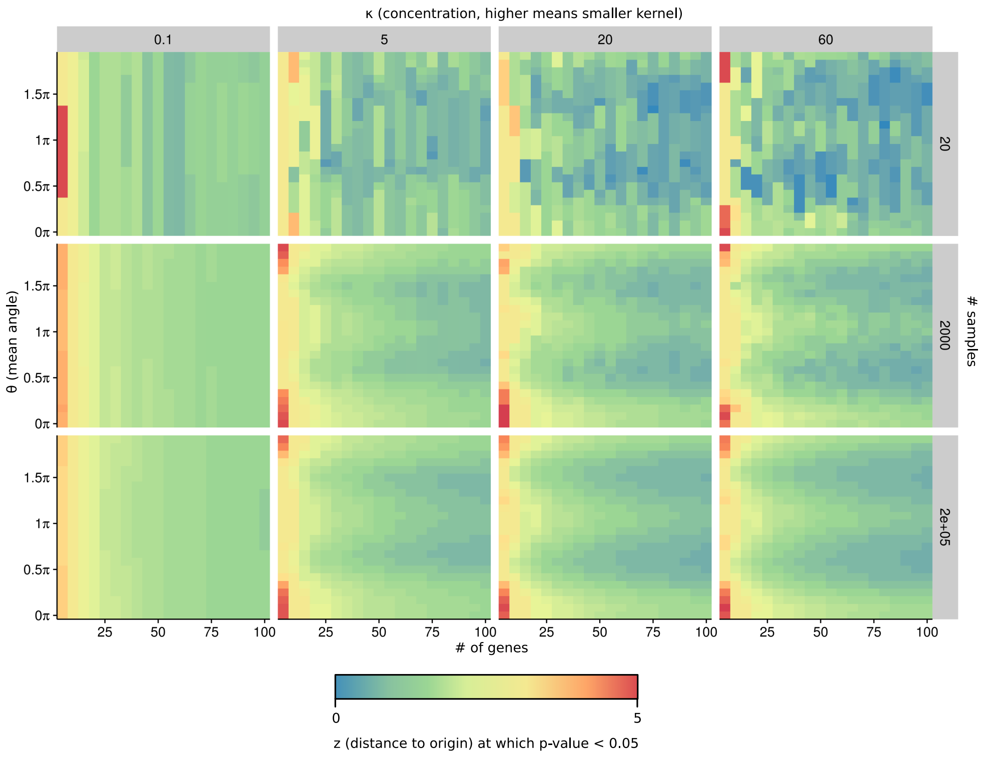


Figure 2.4: Effect of parameters on the estimation of the null distribution. Shown is the z value at which the p-value becomes lower than 0.05 for a particular gene set (of n genes, x-axis) at a particular κ (y-axis) for the dataset in Figure 2.3. Higher concentration values provide a more detailed estimation, but at the cost of requiring more samples.

Applying triwise to analyse macrophage progenitors before and after adoptive transfer

Triwise was initially developed for the analysing the complexity and origin of macrophage progenitors.

A brief introduction to macrophage biology

Macrophages are "jack-of-all-trades" immune cells. On the one hand do they perform many immune functions, such as clearance of pathogens, signalling to innate and adaptive immune cells, and dampening of the immune response [5]. But they are also important in maintaining tissue homeostasis through tissue remodelling, by regulating the viability of cells, clearance of apoptotic cells through their phagocytic ability, and by acting as signalling mediators to other tissue cells [6, 7]. In some cases, they also perform steady-state organ functions themselves, for example by maintaining iron and lipid metabolism in liver and blood [8].

Tissue-resident macrophages are present in most tissues in mammals [7]. Given that each type performs a very specific role in the tissue, it is perhaps not surprising that tissue-resident macrophages have very different transcriptomes and chromatin landscapes [9]. It is also clear that in many mice tissues, such as the brain, alveoli and liver, macrophages are self-maintaining, and thus do not require input from haematopoiesis to persist in the tissue. These tissue-resident macrophages have typically an embryonic origin, coming from yolk-sac macrophages and/or fetal liver monocytes [10]. Other tissues such as the heart, pancreas and gut do require input from the bone marrow for maintaining macrophage populations, under the form of bone marrow monocytes [10].

Experimental approach

There are several possible explanations of why different macrophage subsets have a different origin. It might be that there are differences in accessibility, such as the blood-brain barrier, that does not allow new progenitor cells to colonise the niche. Even if the niche is still accessible, it might also be that other factors coming from the resident macrophages or other tissue cells inhibit the differentiation of new macrophages. If the tissue-resident macrophages have a different origin, might part of the diversity in function and transcriptional profile be explained by a difference in origin? In other words, could it be that part of the explanation of why an alveolar macrophage in adult mice is so different from a dermal macrophage is because the former is derived from fetal monocytes, while the latter develops from bone marrow monocytes?

To analyse this, we developed a mouse model in which neonatal mice do not have alveolar macrophages due to a knock-out in the *Csf2rb* gene. This gene encodes for the β-chain of the GM-CSF receptor, which receives an indispensable signal from GM-CSF to allow fetal liver monocytes to differentiate into alveolar macrophages [11]. These mice were then injected intranasal with macrophage progenitor populations that do have functional *Csf2rb* copies, which allowed us to assess the capacity of each of the progenitors to develop into AMs.

We isolated the three major macrophage progenitor populations (Yolk-sac macrophages, fetal liver monocytes and bone marrow monocytes) through fluorescence assisted cell sorting (FACS) [11]. After intranasal injection in *Csf2rb*$^{-/-}$ mice, the AM populations that originate from the injected population were also sorted through FACS. After RNA extraction and cDNA synthesis, these samples were analysed using Affymetrix Chip Mouse Gene 1.0 ST microarrays.

The Robust Multi-array Average (RMA) procedure, as implemented in the bioconductor affy package [12] was used to normalise data within arrays (probeset summarization, background correction and log2-transformation) and between arrays (quantile normalisation). Only probesets that mapped uniquely to one gene were kept, and for each gene, the probeset with the highest average expression level was kept. PCA plots were created using the 15% of genes with the most variable expression. Differentially expressed genes were identified as Log2 fold change > 1 or < -1, Adjusted P value < 0.01 (limma Bioconductor package ([13], corrected for multiple testing by the Benjamini-Hochberg method).

Triwise application

By visualising the differences between the three progenitor populations through a triwise dotplot, it becomes immediately clear that the main differences in gene expression are between the yolk-sac macrophages and the two monocyte populations (Figure 2.5). To get an intuitive feeling for the triwise dotplots, we highlight 4 example genes. *Ccr2* is specifically expressed in the two monocyte subsets, and thus located between the BM-MO and FL-MO axes. *Mertk* on the other hand is not expressed by monocytes and is thus present directly on the YS-Mac axis. Two genes are highlighted in the middle: a gene encoding a β-tubulin (*Tubb5*) is, as a typical housekeeping gene, highly expressed in all conditions, and *Siglecf*, a gene that is expressed by alveolar macrophages but not by its precursors.

To better understand how each progenitor might intrinsically differ in its capacity to become a functional AM, we analysed the differential expression of functional gene sets among the progenitors. We defined some landmark functions of progenitor cells and AMs: migration, proliferation, alveoli homeostasis, antigen presentation, pathogen clearance and pathogen recognition. For each of these functions we defined functional gene sets by combining related gene ontology term annotations [14].

We found that most functional sets related to anti-bacterial responses, cell movement, immune responses, lipid metabolism, pattern recognition followed the same pattern as all other genes, namely along the monocyte - yolk-sac macrophage axis (Figure 2.5B). Chemokine recognition had a more scattered pattern (Figure 2.5C), which might indicate a different migratory potential for the different progenitors. While this showcases the transcriptional divergence, it does not indicate any clear functional differences between the progenitor populations.

Other functional gene sets did show more coherent patterns. Genes related to anti-viral immune functions were relatively downregulated in fetal liver monocytes (Figure 2.5D). Proliferation genes were strongly upregulated towards the two embryonic cell populations, including both genes related to cell division (*Cdca2* and *Cenph*) and DNA replication (*Mcm4* and *Pola1*) (Figure 2.5E). Genes related to antigen presentation, including several genes encoding major histocompatibility complex 2 subunits, were more highly expressed in bone marrow monocytes (Figure 2.5F). Of those modules listed here, only anti-viral and proliferation gene sets showed a significant non-uniform trend towards a particular direction according to a Rayleigh z-test (Figure 2.5G).

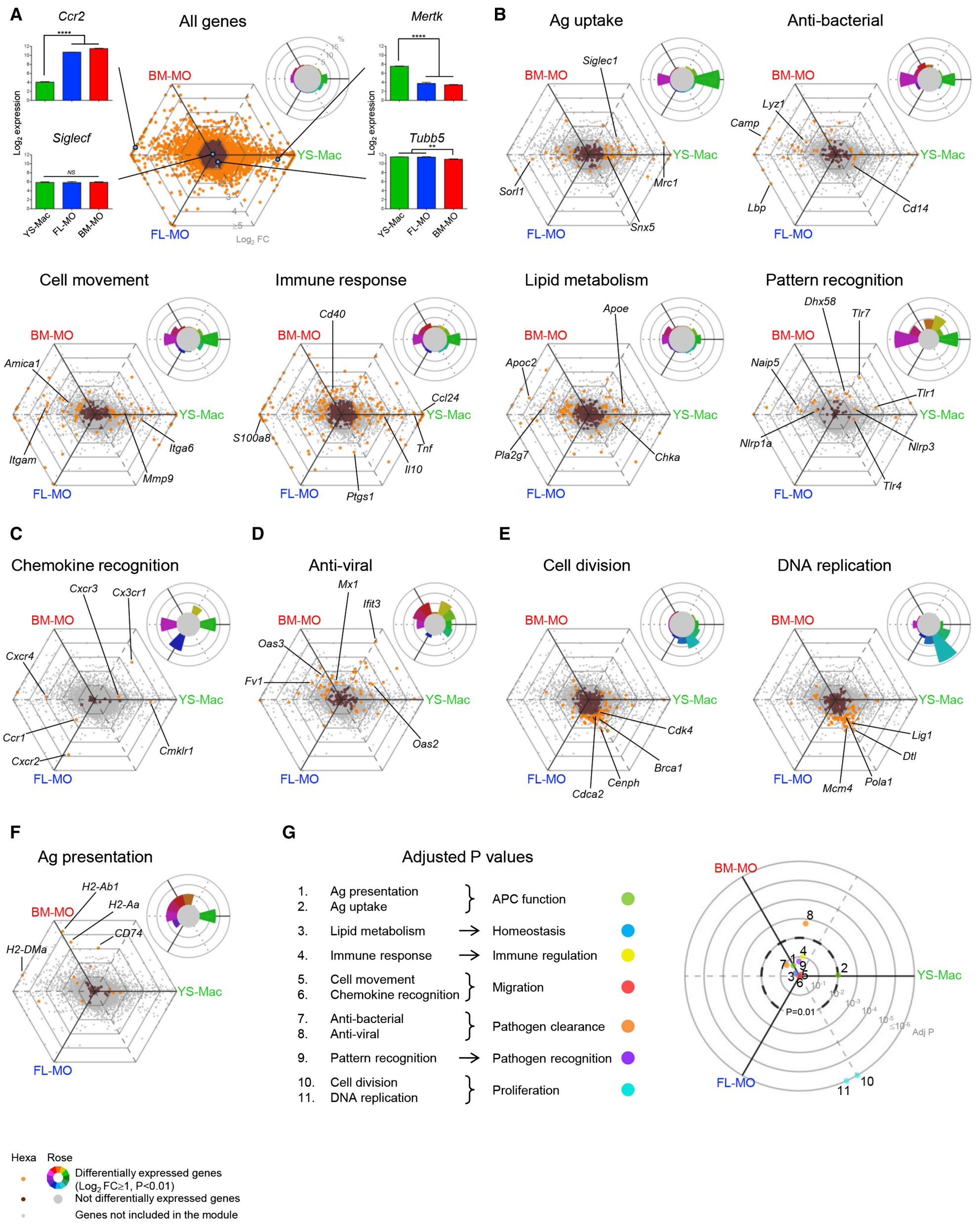


Figure caption on next page →

Figure 2.5: Expression differences between three main macrophage progenitor populations: Yolk-sac macrophages (YS-Mac), Fetal liver monocytes (FL-MO), and Bone marrow monocytes (BM-MO). A Overview dotplot with genes that are differentially expressed in any of the three possible pairwise comparisons highlighted in orange. A monocyte specific gene (*Ccr2*), a Yok-sac macrophage specific gene (*Mertk*), a lowly expressed gene (*SiglecF*) and a housekeeping gene (*Tubb5*). Bar plots show the average log2 normalised expression. The roseplot in the top right indicates the directional distribution of the differentially expressed genes, indicating that most genes are differentially expressed along the monocyte - yolk-sac macrophage axis. **B-F** The differential expression of different functional gene sets related to antigent presentation (APC), alveoli heomeostasis, immune regulation, migration, pathogen clearance, pathogen recognition and proliferation. **: p < 0.01, ****: p < 0.0001, ns: not significant. **G** Statistical significance of unidirectional upregulation of functional gene sets. Each dot represents a gene set, its distance to the origin represents the adjusted p-value (corrected for multiple testing using the benjamini-hochberg method), with the dashed line indicating an false-discovery rate of 0.01 The direction of the dot shows the average direction in which the genes are upregulated. This figure was previously published in van de Laar et al. [1].

Six weeks after adoptive transfer of the progenitors into empty alveolar niches, the expression profiles of the three populations nearly converged (Figure 2.6a). We found that only 22 genes were significantly expressed between any of the AM populations after transfer, primarily up- and downregulated in bone-marrow monocyte and yolk-sac macrophage derived AMs (Figure 2.6b). Although not perfect, it is clear that most of these genes were already differentially expressed towards the same progenitors before transfer (Figure 2.6b). This indicates that the progenitor has only a very limited influence on the expression of the resulting AM, and that all three progenitors are able to differentiate into nearly transcriptionally identical AMs.

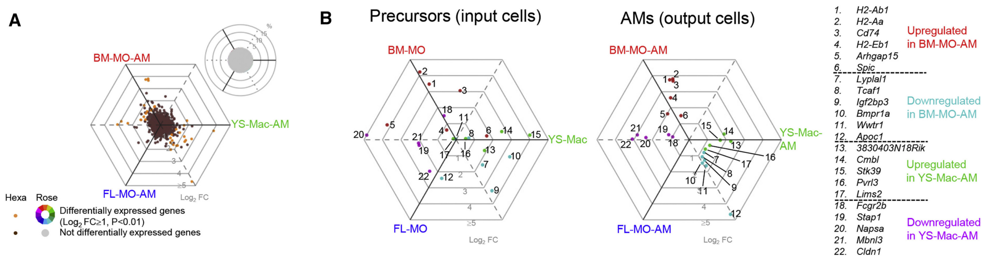


Figure 2.6: Gene expression differences between alveolar macrophage populations coming from three different progenitors. Bone marrow monocyte derived alveolar macrophages (BM-MO-AM), yolk-sac macrophage derived alveolar macrophages (YS-Mac-AM) and fetal liver monocyte derived alveolar macrophages (FL-MO-AM). **A** Transcriptional diversity is reduced to some dozen differentially expressed genes after adoptive transfer. **B** Genes that are still differentially expressed often had a similar expression pattern before transfer. Parts of this figure were previously published in van de Laar et al. [1].

Implications

Could we have reached the same conclusions without triwise analyses? To some extent yes, given that similar results are obtained by dimensionality reduction, differential expression and heatmaps as shown in the paper [1]. Of course, the visualisation onto two dimensions gives a clear and compact way of visualising the heterogeneity between the progenitors, and the absence thereof in the AMs. Where the triwise analysis really shines is with the analysis of functional heterogeneity of the progenitors. This analysis would probably have involved testing functional enrichment between each pairwise comparison, and then testing whether these enriched genes were shared between the pairs. Triwise on the other hand gives a clear and simple representation of the enrichment, with a direction but also a visualisation of the spread of enrichment.

In the same study we showed that in competition, more functional AMs were originating from fetal liver monocytes as compared to the other two progenitors, even when transferred with near equal proportions. The functional gene set analysis might shed some light why this is the case. Perhaps the higher proliferative potential of fetal liver monocytes might give them and edge bone marrow monocytes. At the same time, the higher expression of anti-viral genes in yolk-sac macrophages might indicate some commitment towards an immune state, and conversely a higher plasticity of the fetal liver monocyte.

Although this study was an almost ideal showcase of the triwise methodology, it also shows the main limitation. In the ideal case, Figure 2.6 should have contained four populations: the three progenitors and the wild-type AM. As will be discussed in the discussion at the end of this chapter, this is an often requested feature, but hard to implement in practice.

Other applications

After the initial publication containing triwise, we wrapped the code inside an R package, available at github.com/saeyslab/triwise. This package contains the code to transform into a barycentric coordinate system, visualise this using both dotplots and roseplots, and run gene set enrichment tests using both a competitive and self-contained test.

The package has been used to solve several other biological questions, some of which are already published [15, 16, 17]. I highlight one such study here.

Understanding CD103$^+$CD11b$^+$ dendritic cell development in the gut

In mice, conventional dendritic cells (cDCs) are typically divided in two subsets [18]. cDC1s are XCR1 positive (and usually CD103$^+$) [19], and are specialised in presenting exogenous antigens (cross-presentation) [20] to ultimately elicit a CD8$^+$ T-cell response. cDC2s are defined as CD172a$^+$ (and usually CD11b$^+$) and are important for activating CD4$^+$ T-cells [21].

A unique dendritic cell subset is present in the gut, which is both CD103$^+$ and CD11b$^+$. Because of its dependency on IRF4 [22], an important transcription factor during cDC2 development, this subset has typically been classified as a cDC2. However, it is unclear how it is different from the prototypical cDC2 (CD103$^-$CD11b$^+$), in relation with the prototypical cDC1 (CD103$^+$CD11b$^-$).

To better understand this, the transcriptome of the three different subsets was generated and analysed using triwise dotplots. These indicated that the major transcriptional difference lies between the cDC1 and cDC2 subsets, but with only 61 and 31 genes significantly upregulated in respectively the CD103$^-$CD11b$^+$ and CD103$^+$CD11b$^+$ subsets.

Discussion

Once you wrap your head around the visualisation of genes within triwise plots, it becomes a powerful tool to analyse the diversity of three biological conditions. Nonetheless, there are a couple of issues left that could impact its usefulness and accuracy.

The most requested feature for triwise is the visualisation of more than three conditions. Given that a barycentric transformation reduces the number of dimensions by one, a two dimensional plot will always be limited to three conditions. Although it is possible to visualise four conditions in three dimensions, such a point cloud is hard to interpret and may hide important information (Figure 2.7a). Still, it is possible to extend the functional enrichment tests to many dimensions. This is relatively easy to do for the competitive test, although it may be limited by computational complexity given to estimate the background distribution in multiple dimensions. This may be improved by using some extensions of the Rayleigh z-test that do not need permutations [23].

Plotting fold-changes in two dimensions works well for microarray data, but is not ideal for RNA-seq data. This data contains a strong mean-variance relationship in RNA-seq data [24], with lowly expressed genes typically having higher log fold-changes than highly expressed genes. This results in a lot of genes lying at the

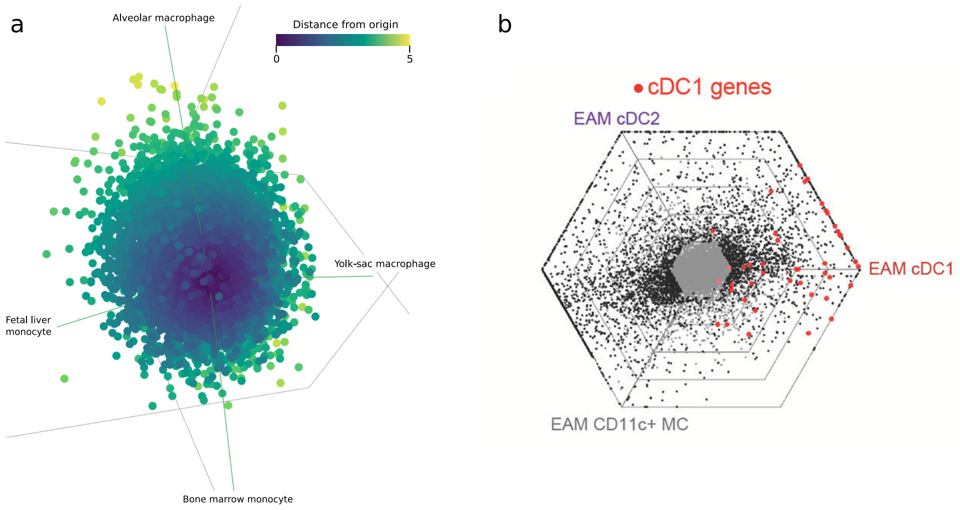


Figure 2.7: Current challenges for the usefulness of triwise. a Visualisation of four conditions is difficult because a three-dimensional is hard to interpret and may hide certain patterns. **b** In RNA-seq data, lowly expressed genes tend to have higher log fold-changes, which makes the typical triwise dotplot harder to interpret. This is an dataset of dendritic cell (DC) subsets during experimental autoimmune myocarditis. Parts of this figure were previously published in Van der Borght et al. [16]

boundaries of the triwise dotplot, despite not being significantly expressed. To resolve this, it might be useful to plot shrunken log-fold changes instead [25].

A final challenge is that the current tests can only detect unidrectional enrichment. Sometimes, it might be useful to detect other patterns as well, such as an upregulation of a gene set in two conditions but not in a third. Although the current test can detect such patterns (such as the anti-viral response in Figure 2.5d), it might not be powerful enough in some cases (such as the antigen presentation genes in Figure 2.5f).

References

[1] Lianne van de Laar et al. “Yolk Sac Macrophages, Fetal Liver, and Adult Monocytes Can Colonize an Empty Niche and Develop into Functional Tissue-Resident Macrophages”. In: *Immunity* 44.4 (Apr. 19, 2016), pp. 755–768. ISSN: 1074-7613. DOI: 10.1016/j.immuni.2016.02.017.

[2] Henryk Maciejewski. “Gene Set Analysis Methods: Statistical Models and Methodological Differences”. In: *Briefings in Bioinformatics* 15.4 (July 2014), pp. 504–518. ISSN: 1477-4054. DOI: 10.1093/bib/bbt002.

[3] K. V. Mardia and Peter E. Jupp. *Directional Statistics.* Wiley, 2000. 472 pp. ISBN: 978-0-471-95333-3.

[4] N. I. Fisher. *Statistical Analysis of Circular Data.* Cambridge University Press, Oct. 12, 1995. 300 pp. ISBN: 978-0-521-56890-6.

[5] Thomas A. Wynn, Ajay Chawla, and Jeffrey W. Pollard. “Macrophage Biology in Development, Homeostasis and Disease”. In: *Nature* 496.7446 (Apr. 2013), pp. 445–455. ISSN: 1476-4687. DOI: 10.1038/nature12034.

[6] Chen Varol, Alexander Mildner, and Steffen Jung. “Macrophages: Development and Tissue Specialization”. In: *Annual Review of Immunology* 33 (2015), pp. 643–675. ISSN: 1545-3278. DOI: 10.1146/annurev-immunol-032414-112220.

[7] Yasutaka Okabe and Ruslan Medzhitov. “Tissue Biology Perspective on Macrophages”. In: *Nature Immunology* 17.1 (Jan. 2016), pp. 9–17. ISSN: 1529-2916. DOI: 10.1038/ni.3320.

[8] Charlotte L. Scott and Martin Guilliams. “The Role of Kupffer Cells in Hepatic Iron and Lipid Metabolism”. In: *Journal of Hepatology* 69.5 (Nov. 1, 2018), pp. 1197–1199. ISSN: 0168-8278, 1600-0641. DOI: 10.1016/j.jhep.2018.02.013.

[9] Yonit Lavin et al. “Tissue-Resident Macrophage Enhancer Landscapes Are Shaped by the Local Microenvironment”. In: *Cell* 159.6 (Dec. 4, 2014), pp. 1312–1326. ISSN: 1097-4172. DOI: 10.1016/j.cell.2014.11.018.

[10] Florent Ginhoux and Martin Guilliams. “Tissue-Resident Macrophage Ontogeny and Homeostasis”. In: *Immunity* 44.3 (Mar. 15, 2016), pp. 439–449. ISSN: 1074-7613. DOI: 10.1016/j.immuni.2016.02.024.

[11] Martin Guilliams et al. “Alveolar Macrophages Develop from Fetal Monocytes That Differentiate into Long-Lived Cells in the First Week of Life via GM-CSF”. In: *Journal of Experimental Medicine* 210.10 (Sept. 23, 2013), pp. 1977–1992. ISSN: 0022-1007, 1540-9538. DOI: 10.1084/jem.20131199.

[12] Laurent Gautier et al. “Affy—Analysis of Affymetrix GeneChip Data at the Probe Level”. In: *Bioinformatics* 20.3 (Feb. 12, 2004), pp. 307–315. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btg405.

[13] Matthew E. Ritchie et al. “Limma Powers Differential Expression Analyses for RNA-Sequencing and Microarray Studies”. In: *Nucleic Acids Research* 43.7 (Apr. 20, 2015), e47–e47. ISSN: 0305-1048. DOI: 10.1093/nar/gkv007.

[14] The Gene Ontology Consortium. “The Gene Ontology Resource: 20 Years and Still GOing Strong”. In: *Nucleic Acids Research* 47.D1 (Jan. 8, 2019), pp. D330–D338. ISSN: 1362-4962. DOI: 10.1093/nar/gky1055.

[15] Charlotte L. Scott et al. “The Transcription Factor Zeb2 Regulates Development of Conventional and Plasmacytoid DCs by Repressing Id2”. In: *The Journal of Experimental Medicine* 213.6 (May 30, 2016), pp. 897–911. ISSN: 1540-9538. DOI: 10.1084/jem.20151715.

[16] Katrien Van der Borght et al. “Myocarditis Elicits Dendritic Cell and Monocyte Infiltration in the Heart and Self-Antigen Presentation by Conventional Type 2 Dendritic Cells”. In: *Frontiers in Immunology* 9 (Nov. 21, 2018). ISSN: 1664-3224. DOI: 10.3389/fimmu.2018.02714.

[17] C. C. Bain et al. “TGFβR Signalling Controls CD103+CD11b+ Dendritic Cell Development in the Intestine”. In: *Nature Communications* 8 (Sept. 20, 2017). ISSN: 2041-1723. DOI: 10.1038/s41467-017-00658-6.

[18] Martin Guilliams et al. “Dendritic Cells, Monocytes and Macrophages: A Unified Nomenclature Based on Ontogeny”. In: *Nature Reviews Immunology* 14.8 (Aug. 2014), pp. 571–578. ISSN: 1474-1741. DOI: 10.1038/nri3712.

[19] Martin Guilliams et al. “Unsupervised High-Dimensional Analysis Aligns Dendritic Cells across Tissues and Species”. In: *Immunity* 45.3 (Sept. 20, 2016), pp. 669–684. ISSN: 1074-7613. DOI: 10.1016/j.immuni.2016.08.015.

[20] Olivier P. Joffre et al. “Cross-Presentation by Dendritic Cells”. In: *Nature Reviews Immunology* 12.8 (Aug. 2012), pp. 557–569. ISSN: 1474-1741. DOI: 10.1038/nri3254.

[21] S. C. Eisenbarth. “Dendritic Cell Subsets in T Cell Programming: Location Dictates Function”. In: *Nature Reviews Immunology* 19.2 (Feb. 2019), p. 89. ISSN: 1474-1741. DOI: 10.1038/s41577-018-0088-1.

[22] Andreas Schlitzer et al. “IRF4 Transcription Factor-Dependent CD11b+ Dendritic Cells in Human and Mouse Control Mucosal IL-17 Cytokine Responses”. In: *Immunity* 38.5 (May 23, 2013), pp. 970–983. ISSN: 1097-4180. DOI: 10.1016/j.immuni.2013.04.011.

[23] Alize E H Scheenstra et al. “The 3D Moore-Rayleigh Test for the Quantitative Groupwise Comparison of MR Brain Images”. In: Information Processing in Medical Imaging : Proceedings of the ... Conference. Vol. 21. Feb. 1, 2009, pp. 564–75. DOI: 10.1007/978-3-642-02498-6_47.

[24] Charity W. Law et al. “Voom: Precision Weights Unlock Linear Model Analysis Tools for RNA-Seq Read Counts”. In: *Genome Biology* 15.2 (Feb. 3, 2014), R29. ISSN: 1474-760X. DOI: 10.1186/gb-2014-15-2-r29.

[25] Michael I Love, Wolfgang Huber, and Simon Anders. “Moderated Estimation of Fold Change and Dispersion for RNA-Seq Data with DESeq2”. In: *Genome Biology* 15.12 (2014). ISSN: 1465-6906. DOI: 10.1186/s13059-014-0550-8.

3 | Comparing module detection methods

Already since my master's thesis, I have been working on methods that try to find modules of co-expressed genes within gene expression data. Clustering methods are most often used here, but a lot of alternatives such as biclustering have been proposed. These have some convincing theoretical advantages, such the ability to detect more local expression changes in a subset of samples (or cells). Despite these, I found it difficult to find good use cases of these methods on our own datasets. Moreover, I found only limited use cases of these methods in literature; most research was focused on development of new methods and not on their application. That's why we set out to find out why these methods were apparently so difficult to use, and provide some guidelines for the field to promote their use. This resulted in a large benchmarking study, that is presented in this chapter.

I have embedded the most relevant supplementary material directly within this thesis, including Supplementary Note 1. Other supplementary material is available at https://www.nature.com/articles/s41467-018-03424-4#Sec19.

At the end of this chapter, I discuss some issues that have popped up after the publication of this paper (section Update).

A comprehensive evaluation of module detection methods for gene expression data

Published in Nature Communications, March 15 2018, doi.org/gc9x36
Wouter Saelens, Robrecht Cannoodt, Yvan Saeys

Contributions
Designed the study: **Wouter Saelens** and Yvan Saeys
Performed the study: **Wouter Saelens**
Supervised the work: Yvan Saeys
Wrote the paper: **Wouter Saelens**, Robrecht Cannoodt and Yvan Saeys

Abstract
A critical step in the analysis of large genome-wide gene expression datasets is the use of module detection methods to group genes into co-expression modules. Because of limitations of classical clustering methods, numerous alternative module detection methods have been proposed, which improve upon clustering by handling co-expression in only a subset of samples, modelling the regulatory network, and/or allowing overlap between modules. In this study we use known regulatory networks to do a comprehensive and robust evaluation of these different methods. Overall, decomposition methods outperform all other strategies, while we do not find a clear advantage of biclustering and network inference-based approaches on large gene expression datasets. Using our evaluation workflow, we also investigate several practical aspects of module detection, such as parameter estimation and the use of alternative similarity measures, and conclude with recommendations for the further development of these methods.

Introduction

Ever since the introduction of genome-wide gene expression profiling technologies, module detection methods have been a cornerstone in the biological interpretation of large gene expression compendia [1, 2, 3]. Modules in this context are defined as groups of genes with similar expression profiles, which also tend to be functionally related and co-regulated. Apart from allowing a more global and objective interpretation of gene expression data [4, 5], co-expression modules are also frequently used to infer regulatory relationships between transcription factors and putative target genes [6, 7, 8]. In addition, modules can improve functional genome annotation through the guilt-by-association principle [9] and allow a better understanding of disease origin [10] and progression [11].

Numerous approaches and algorithms have been proposed for module detection in gene expression data. The most popular approach, clustering, has been used since the first gene expression datasets became available and is still the most widely used to this day [6, 7, 8, 10]. However, in the context of gene expression, clustering methods suffer from three main drawbacks. First, clustering methods only look at co-expression among all samples. As transcriptional regulation is highly context specific[12], clustering potentially misses local co-expression effects which are present in only a subset of all biological samples. Second, most clustering methods are unable to assign genes to multiple modules. The issue of overlap between modules is especially problematic given the increasing evidence that gene regulation is highly combinatorial and that gene products can participate in multiple pathways [13, 14]. A third limitation of clustering methods is that they ignore the regulatory relationships between genes. As the variation in target gene expression can at least be partly explained by variation in transcription factor expression [15], including this information could therefore boost module detection. Several alternative module detection approaches have therefore been developed in order to alleviate these three limitations. Decomposition methods [16] and biclustering [17] try to handle local co-expression and overlap. These methods differ from clustering because they allow that genes within a module do not need to be co-expressed in all biological samples, but that a sample can influence the expression of a module to a certain degree (decomposition methods) or not at all (biclustering methods). Two other alternative methods, direct network inference [15] (direct NI) and iterative NI [18], use the expression data to additionally model the regulatory relationships between the genes.

Given the importance of module detection within the transcriptomics field and the wealth of available methods, it is critical that existing and new approaches are evaluated on objective benchmarks. However, we identified several shortcomings in past evaluation studies [17, 19, 20, 21, 22], related to the use of multiple evaluation metrics, the correct tuning of parameters, and the biological relevance of synthetic data. In this study we therefore propose a new evaluation pipeline for module detection methods for gene expression data. Central to our approach is that we use known regulatory networks to define sets of known modules, which can be used to directly assess the sensitivity and specificity of the different module detection methods on real data. Using our evaluation strategy we analyse the performance of 42 module detection methods spanning all five main approaches. We also consider several practical aspects of module detection, such as the relative data requirements of the methods, parameter estimation, and the use of alternative similarity measures for clustering. The purpose of this evaluation study is twofold. We first want to provide an overview of the characteristics and performance of current module detection methods to guide the biologist in their choice. Second, we propose a benchmark strategy, which can be used in future studies to compare novel methods with the current state of the art. For this purpose, we provide all gold standards, expression datasets, and the evaluation procedure to the community.

Results

Evaluation workflow

Our evaluation procedure was structured as follows (Figure 3.1). We applied publicly available module detection methods on nine gene expression compendia from *Escherichia coli*, yeast, human, and *in silico* simulated regulatory networks (Figure 3.1a). We scored the different methods by comparing the observed modules with a set of known modules. These known modules were extracted from known regulatory networks using three different module definitions (Figure 3.1b), two requiring co-regulation by either one or all known regulators and one looking at strong interconnectedness within the gene regulatory network. To compare a set of observed modules with known modules, we considered several scores described in the literature (Supplementary Note 1) and ultimately chose four scores as follows: recovery, relevance, recall, and precision (Figure 3.2). Note that classical scores comparing clusterings could not be used because these cannot handle overlap. As all methods generally performed equally or worse than random on human datasets, due to the high number of false positives in the gold standard (Supplementary Note 1), we instead used a scoring system which looks at how well the observed modules cover the targets of regulators in the dataset (Figure 3.1e). To avoid certain gold standards and module definitions from disproportionately influencing our final score, we normalised each score using random permutations of the known modules. The final score for a method ultimately represented a fold improvement of a given module detection method over the score obtained from randomly permuted known modules.

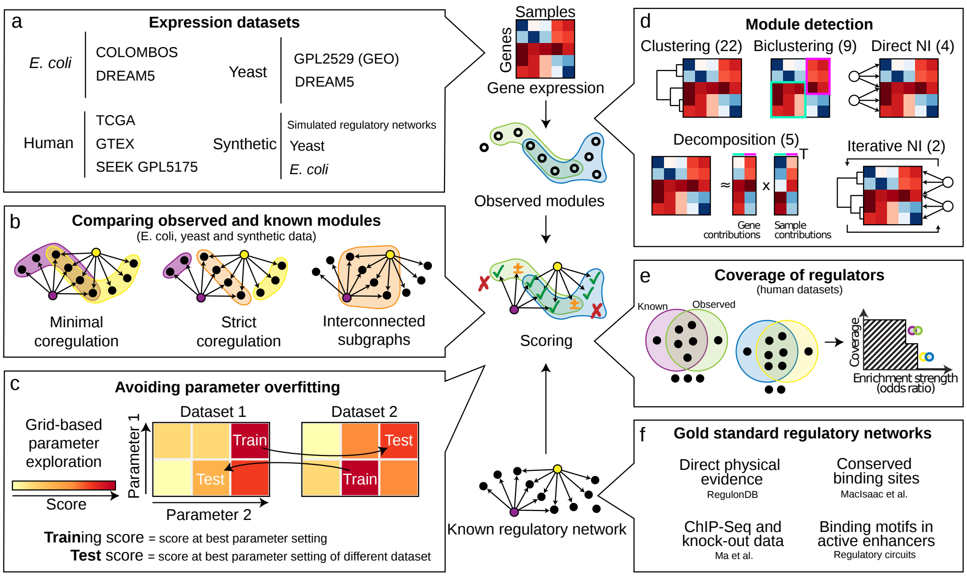


Figure 3.1: Overview of our evaluation methodology. **a)** The nine different datasets used in this evaluation. **b)** We used three different module definitions to extract known modules from known regulatory networks for the evaluation on *E. coli*, yeast and synthetic data. **c)** To avoid parameter overfitting on characteristics of particular datasets, we first optimised the parameters on every dataset using a grid search, and then used the optimal parameters on one dataset (training score) to assess the performance of a method on another dataset (test score). **d)** We evaluated a total of 42 methods, which can be classified in 5 categories: clustering, biclustering, direct network inference (NI), decomposition, and iterative NI. **e)** For the evaluation on human data, we compared how well the targets of each regulator is enriched in at least one of the modules. **f)** We used four different regulatory networks in our evaluation, each generated from different types of data.

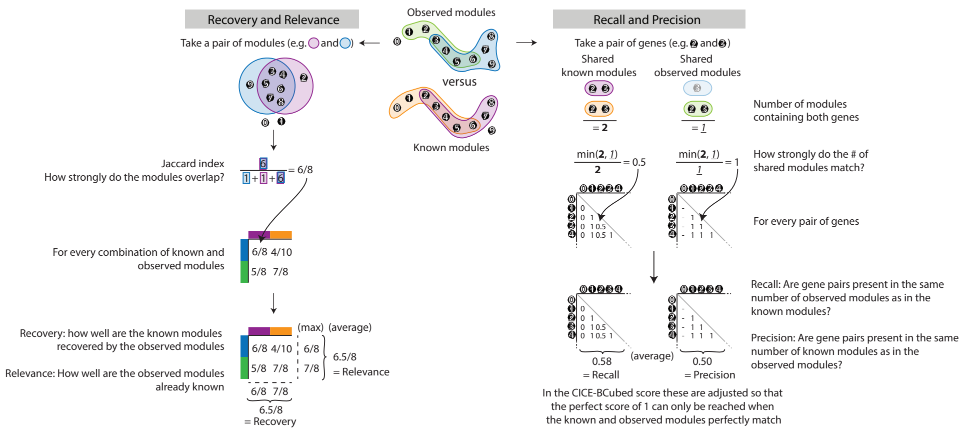

Recovery and Relevance
Take a pair of modules (e.g. and)
Jaccard index
How strongly do the modules overlap?
= 6/8
For every combination of known and observed modules
Recovery: how well are the known modules recovered by the observed modules
Relevance: How well are the observed modules already known
(max) (average)
6.5/8
= Relevance
6.5/8
= Recovery
Observed modules
versus
Known modules
Recall and Precision
Take a pair of genes
Shared known modules
Shared observed modules
Number of modules containing both genes
How strongly do the # of shared modules match?
For every pair of genes
Recall: Are gene pairs present in the same number of observed modules as in the known modules?
Precision: Are gene pairs present in the same number of known modules as in the observed modules?
0.58
= Recall
(average)
0.50
= Precision
In the CICE-BCubed score these are adjusted so that the perfect score of 1 can only be reached when the known and observed modules perfectly match

Figure 3.2: Illustration of the four main scores used in this study.
Each score assesses the similarity between a set of known modules and a set of observed modules. The recovery and relevance will try to match individual modules between the two sets using the Jaccard index, a measure of overlap between two mathematical sets. The recovery tries to match known modules with observed modules, while the relevance tries to match observed modules with known modules. The recall and precision scores will compare the number of times a pair of genes is together present in observed modules versus those in known modules. The recall determines whether a pair of genes is present in at least the same number of modules in the known modules as in the observed modules, and vice-versa for the precision.

Parameter tuning is a necessary but often overlooked challenge with module detection methods. Although good performance generally depends on the correct choice of parameters, this also increases the risk of overfitting on specific characteristics of one dataset, as such parameters will lead to suboptimal results when generalizing the parameter settings to other datasets. To address both problems, we optimised the parameters for every method with a grid search (Supplementary Note 2) and used an approach akin to cross-validation where the optimal parameter settings from one dataset were used to assess the performance of a method on another dataset (Figure 3.1c). For every method we give two scores: the training score represents the score at the optimal parameter settings, whereas the test score denotes the performance when parameters were estimated on an alternative dataset (Figure 3.1c).

Overall performance

We evaluated a total of 42 module detection algorithms covering all 5 approaches (clustering, decomposition, biclustering, direct NI, and iterative NI) using the described methodology (Table 3.1 and Supplementary Note 2). Overall, our results indicate that decomposition methods detect the modules which best correspond to the known modular structure within the gene regulatory network (Figure 3.3a). The best decomposition methods are all variations of independent component analysis (ICA) with different post-processing methods [16, 23]. Surprisingly, neither biclustering nor direct NI, nor iterative NI methods outperform clustering methods, although in theory they should offer several advantages by allowing overlap, modelling transcriptional regulation and/or looking for local co-expression effects (Figure 3.3b).

	Clustering: grouping genes based on a global similarity in gene expression profiles
A	*FLAME*: fuzzy clustering by selecting cluster supporting objects based on the K-nearest neighbour density estimation
B	*K-medoids*: iteratively refines the centers (which are individual genes) and the average dissimilarity within the cluster
C	*K-medoids* (see B) but with automatic module number estimation
D	*Fuzzy c-means*: similar to k-means (see F), but using fuzzy instead of crisp cluster memberships
E	*Self-organizing maps*: maps each gene on a node embedded in a two dimensional graph structure
F	*K-means*: iteratively refines the mean expression with a cluster and the within-cluster sum of squares
G	*MCL*: simulates random walks within the co-expression graph by alternative steps of expansion and inflation
H	*Spectral clustering*: applies K-means in the subspace defined by the eigenvectors of the Pearson correlation affinity matrix
I	*Affinity propagation*: clustering by exchange of messages between genes
J	*Spectral clustering*: applies K-means in the subspace defined by the eigenvectors of the K-nearest neighbour graph
K	*Transitivity clustering*: tries to find the transitive co-expression graph in which the total cost of added and removed edges is minimised
L	*WGCNA*: agglomerative hierarchical clustering (see M), but using the topological overlap measure and a dynamic tree cutting algorithm to implicitly determine the number of modules
M	*Agglomerative hierarchical clustering*: generates a hierarchical structure by progressively grouping genes and clusters based on their similarity
N	*Hybrid hierarchical clustering*: combination of agglomerative and divisive hierarchical clustering
O	*Divisive hierarchical clustering*: generates a hierarchical structure by progressively splitting the genes into clusters
P	Agglomerative hierarchical clustering (see M), but with automatic module number estimation
Q	*SOTA*: combination of self-organizing maps and divisive hierarchical clustering
R	First finds cluster centers by searching for high density, each gene is then assigned to the cluster of its nearest neighbour of higher density
S	*CLICK*: uses density estimation to find tight groups of similar genes, after which these are expanded into modules
T	*DBSCAN*: groups genes within core, non-core and outlier genes based on the number of nearest neighbours

U	*Clues*: first applies a shrinking procedure which moves each gene towards nearby high density regions, after which the genes are partitioned into an automatically determined number of clusters using the silhouette width
V	*Mean shift*: moves each gene towards nearby high density regions until convergence
	Decomposition: extracting the components corresponding to co-expression modules by decomposing the expression matrix in a product of smaller matrices
A	*Independent component analysis*: decomposes the expression matrix into a set of independent components using the FastICA algorithm, detects potentially overlapping modules within each source signal using false-discovery rate (FDR) estimation
B	Similar to A, but detects two modules per independent component depending on whether genes have positive or negative weights
C	Similar to A, but detects modules within each source signal using z-scores
D	Combination of principal component analysis and independent component analysis, uses FDR estimation to find modules
E	*Principal component analysis*: decomposes the expression matrix into a set of linearly uncorrelated components, detects potentially overlapping modules within each component using FDR estimation
	Biclustering: simultaneous grouping of genes and samples in biclusters based on similar local behavior in expression
A	*Spectral biclustering*: detecting checkerboard patterns within the gene expression matrix
B	*ISA*: iteratively refines a set of genes and samples based on high or low expression in both the gene and sample dimension
C	*QUBIC*: finds biclusters in which the genes have similar high or low expression levels in a discretised expression matrix
D	*Bi-Force*: finds biclusters with over- or under-expression by solving the bicluster editing problem
E	*FABIA*: builds a multiplicative model of the expression matrix layer by layer. Every layer represents a bicluster
F	*Plaid*: builds an additive model of the expression matrix layer by layer. Every layer represents a bicluster
G	*MSBE*: finds additive biclusters starting from randomly sampled reference genes and conditions
H	*Cheng & churc*h: minimises the mean squared residue within every bicluster
I	*OPSM*: searches for biclusters where the expression changes in the same direction between genes and samples
	Iterative network inference: iterative optimisation of an inferred network and a set of clusters
A	*MERLIN*: iteratively refines a direct regulatory network and modules within a probabilistic graphical network framework
B	*Genomica*: starts from an initial hierarchical clustering and iteratively refines this clustering and an inferred module network using a model based on Bayesian regression trees
	Direct network inference: inference of a regulatory network based on gene expression similarity between regulators and target genes
A	*GENIE3*: predicts the expression of each target gene based on random forest regression

B *CLR*: calculates the likelihood of mutual information estimations based on the network neighbourhood
C *Pearson's correlation* between regulator and target gene
D *TIGRESS*: network inference using a combination of Lasso sparse regression and stability selection

Table 3.1: Module detection methods evaluated in this study Within each category, methods are ranked according to their average test score (Figure 3.3). We refer the reader to Supplementary Note 2 for details regarding the implementation and parameters

Note that decomposition methods not only perform well when the gold standard modules contains overlap, in the case of minimally co-regulated modules, but also when no overlap is present in the known modules (Figure 3.3d). To further investigate this, we calculated separate scores for genes within one or multiple modules. This analysis showed that both clustering and decomposition methods are better at grouping genes that are present in one module, whereas biclustering and direct NI methods are slightly biased toward genes present in more than one module (Supplementary Fig. 2). These results indicate that the higher performance of decomposition methods over clustering is not exclusively caused by their ability to detect overlapping modules, but that also other factors such as local co-expression could have a role.

We further classified clustering algorithms into four categories: graph-based clustering, representative-based clustering, hierarchical clustering, and density-based clustering. We found that graph-based, representative-based, and hierarchical clustering all performed equally well, with the clustering method FLAME (Fuzzy clustering by Local Approximation of Memberships) [24], one of the only clustering methods able to detect overlap, slightly outperforming other clustering methods. Among hierarchical clustering methods, agglomerative methods provide the highest performance compared with the intermediate and low scores of respectively hybrid and divisive methods. Density-based clustering methods on the other hand had much lower performance, which can be partly explained by a higher parameter sensitivity for some density-based methods. Although the overall performance of biclustering methods was low, we also made a similar categorization of these methods based on the type of biclusters they detect. Methods that detect constant or extreme biclusters generally outperformed other methods detecting more complex bicluster patterns. In fact, except for FABIA (Factor Analysis for Bicluster Acquisition), the performance of the latter methods was generally not much better or even worse than random permutations of the known modules.

Although the relative ranking of the main methods is remarkably stable across datasets (Figure 3.3c and Supplementary Fig. 3a), individual methods can still perform well in one setting even though their overall performance is poor (Supplemen-

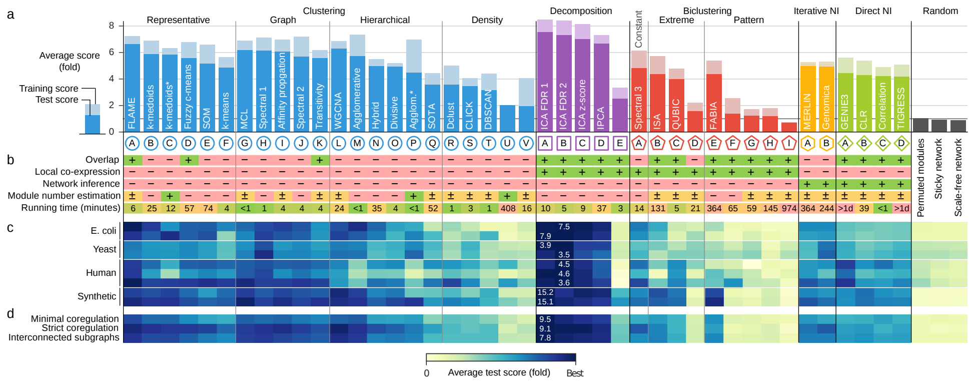


Figure 3.3: Overall performance of 42 module detection methods (Table 3.1)) based on the agreement between observed modules and known modules in gene regulatory networks. The methods can be divided in five categories: clustering, decomposition, biclustering, direct network inference (direct NI) and iterative network inference (iterative NI) methods. Clustering and biclustering methods were further classified in subcategories (see Methods). **a** Average test and training scores across datasets and module definitions. The score represents a fold improvement over permutations of the known modules. *Automatic estimation of number of modules. **b** Different properties of the module detection methods (see Supplementary Note 2). A + (green background) denotes that a method can handle a certain property listed on the left. We distinguish between explicit μ (−), implicit (±), and automatic (+) module number estimation. Note that running times strongly depend on the implementation, hardware, dataset dimensions, and parameter settings, and are therefore only indicative. **c** Test scores at each of the four datasets, averaged over module definitions. **d** Test scores on each of the three module definitions, averaged over different datasets

tary Fig. 4). Most profoundly is the higher performance of certain biclustering methods, such as ISA (Iterative Signature Algorithm), QUBIC (Qualitative Biclustering), and FABIA, and direct NI methods, primarily GENIE3, on human and/or synthetic data, where these methods can in some cases compete with clustering and decomposition methods. Performance was generally very consistent across different module definitions (Figure 3.3d and Supplementary Fig. 3b), despite limited similarity between the sets of known modules (Supplementary Fig. 5). We also found that the overall ranking of the methods remained similar when we used different randomization procedures to normalise our scores (Supplementary Fig. 6). The relative performance of individual methods was more variable when we compared different scores, especially for scores which can handle overlapping modules, although the overall ranking of the different module detection approaches remained stable (Figure 3.4 and Supplementary Fig. 8). Together, this again highlights the importance of using multiple datasets and scoring metrics for a robust and unbiased evaluation of bioinformatics methods [25, 26].

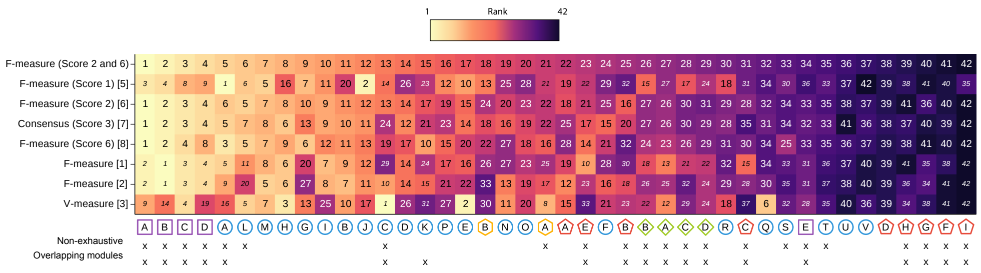


Figure 3.4: Comparison of method ranks across different scoring metrics.
Each score (y-axis) assesses the correspondence between a set of observed and known modules. Some scores have some difficulties with handling overlapping and/or non-exhaustive module assignment. Ranks which are potentially unreliable, if also the method detects non-exhaustive and/or overlapping modules (bottom), are therefore shown in a smaller font. Despite this, we found that the overall ranking of most methods was similar between most scores.

Parameter tuning

The need to tune parameters on individual datasets varied greatly among methods, which we quantified by comparing training and test scores. Some methods, such as FLAME, WGCNA (Weighted Gene Co-expression Network Analysis), and MERLIN (Modular regulatory network learning with per gene information) were relatively insensitive to parameter tuning, despite requiring the optimisation of two or more parameters (Figure 3.3a). On the other hand, methods such as fuzzy c-means, self-organizing maps, and agglomerative hierarchical clustering performed very differently between test and training parameters. Nonetheless, the overall ranking of the different module detection approaches does not change drastically between training and test scores. The top decomposition methods for instance outperform clustering methods both before and after controlling for parameter overfitting (Figure 3.3a and Supplementary Fig. 4).

The most central parameters in module detection (and unsupervised data analysis in general) are those affecting the number of clusters detected within a dataset. We distinguish three different ways a method determines the number of modules (Figure 3.3b). Explicit methods, such as k-means and all decomposition methods, require that the number of modules is specified by the user. Implicit methods, such as affinity propagation, adapt the module number on each dataset based on other parameters supplied by the user. Finally, automatic methods determine the number of modules completely automatically, usually by iterating over several parameter settings and selecting the one that optimises some criterion of cluster quality. Measures for cluster quality can range from the stability of clusters among several resamplings of the dataset [27], the balance between cluster tightness and separateness (as measured by cluster validity indices [28]), or the optimal functional enrichment of the modules [22]. Although the top method within each clustering subcategory estimate the number of modules implicitly (coinciding with a relatively low parameter sensitivity) (Figure 3.3b), we found that implicit or automatic estimation of the number of modules is not a prerequisite for a high performance (Figure 3.5). Indeed, the top decomposition methods all require the number of modules to be specified beforehand. Interestingly, the performance of iterative NI methods depended only little on the initial parameters, possibly because these methods adapt the number of modules depending on the inferred regulatory network.

Apart from those parameters influencing the number of clusters, most module detection methods also have other parameters, frequently affecting the compactness of the modules, the way local co-expression is defined or the minimal number of genes within a module. As all these parameters can have significant effects on the resulting modules (and thus the performance of a method), we assessed how well automatic parameter estimation can improve method performance. Automatic parameter estimation can be seen as an alternative to determining the optimal parameters on one or more training datasets and using these parameters on a test dataset to assess performance. Based on a previous evaluation study [28] we chose four of the most promising cluster validity indices, and found that the benefits of cluster validity indices were mostly confined to clustering methods (Figure 3.6). Notably, spectral clustering, affinity propagation, and k-medoids frequently increased in test score when their parameters were automatically estimated using the average silhouette width and the Davis–Bouldin index (Supplementary Fig. 10). On the other hand, the performance of FLAME clustering and decomposition methods generally decreased when using cluster validity indices, usually performing even worse than randomly selecting parameters within the parameter grid (Supplementary Fig. 11).

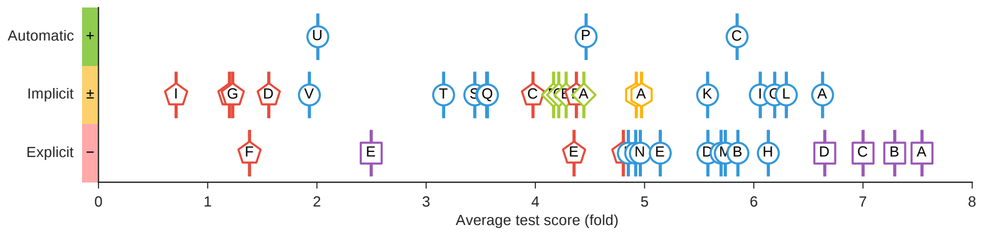


Figure 3.5: Comparison between different ways of estimating the number of modules. One of the most important parameters for most module detection methods are those influencing the number of modules found in the dataset. There are three ways a method can determine the number of modules: (i) explicitly, by retrieving a fixed number of modules determined by the user, (ii) implicitly, by using other parameters determined by the user to estimate the number of modules and (iii) automatically, by determining the number of modules independent of parameters. Implicitly or automatically determining the number of methods can therefore allow methods to better adapt to individual characteristics of a dataset, although it can also lead to a suboptimal number of modules when compared with a given gold standard. We found that among clustering methods those that implicitly estimated the number of methods performed better than their explicit counterparts. However, implicit or automatic module number estimation is (among current methods) not mandatory for optimal performance, as all decomposition methods have a user parameter for the number of components (and thus the number of modules) within the data.

It is important to note here that all four cluster validity indices have been developed in the context of clustering methods and were therefore never really designed with overlap and local co-expression in mind, which could explain their low performance with these methods. We also analysed two alternative measures, which try to estimate the number of clusters based on the optimal enrichment of functional terms or pathways within the modules. We found that a measure that assesses both the coverage of all functional terms as well as the strong individual enrichment of every module (F-aucodds) performed very well, in a large majority of cases performing better than using the optimal parameters of another dataset (Supplementary Fig. 10) and random parameter settings (Supplementary Fig. 11).

Another important parameter for most clustering methods is the distance or similarity measure for comparing gene expression profiles. The most popular metric for gene expression data is undoubtedly the Pearson's correlation coefficient, which measures the extent of the linear dependence between two expression profiles regardless of differences in absolute expression levels. Several authors have criticised this measure [29, 30, 31], mainly due to three limitations: (i) it ignores inverse relations between genes (Figure 3.7a), (ii) it is unable to capture non-linear relationships (Figure 3.7b), and (iii) it is not robust to outliers and skewed distributions (Figure 3.7c). Several alternative measures have therefore been proposed, which try to tackle some of these limitations (Supplementary Note 3).

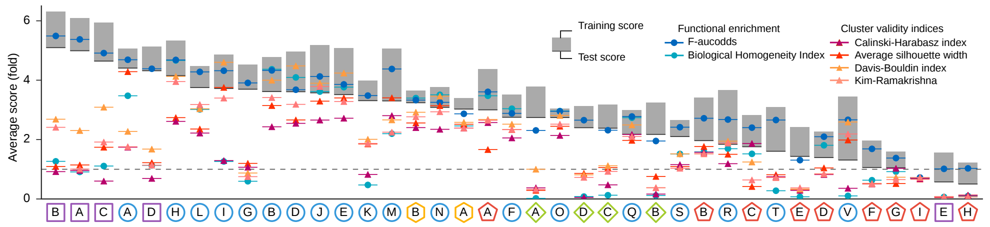


Figure 3.6: Effect of automatic parameter estimation using four different cluster validity indices and two measures based on functional enrichment on the performance of top module detection methods. Shown are changes in test scores after parameter estimation (either using measures based on functional enrichment in blue or cluster validity indices in red–orange), averaged over datasets and module definitions, of the top module detection methods in every category

---

To investigate whether these alternatives are able to improve the module detection, we used 15 such measures as the input for four of the top clustering methods which require having a similarity or distance measure as parameter. Surprisingly, none of the alternative similarity metrics are able to improve performance of any of the four top clustering methods (Figure 3.7d). When investigating this further, we found several cases where these alternative measures can indeed retrieve known co-regulated genes which were ranked lower than Pearson's correlation, as illustrated with three case examples (Figure 3.7a–c, e). However, when comparing the top 10% gene pairs between Pearson's correlation and alternative measures, more known co-regulated gene pairs are removed than there are gained (Figure 3.7f).

Sensitivity to number of samples and noise

We next tested the influence of the number of samples within an expression dataset on the relative performance of the top module detection methods within every category. Although, as expected, the performance of every method declined with decreasing dataset size, the magnitude and timing of this decline varied strongly per method. Notably, ICA-based decomposition methods (decomposition methods A and B) seem to be much more sensitive to the number of samples in the dataset compared with other methods (Figure 3.8 and Supplementary Fig. 13). On the other hand, the performance of several network inference based methods, such as Genomica (iterative NI method A) and GENIE3 (direct NI method A), remained relatively stable with decreasing number of samples. Together, this indicates that despite its better performance on large datasets, current matrix decomposition methods are unable to meet the performance of clustering methods when a smaller number of biological conditions are being considered.

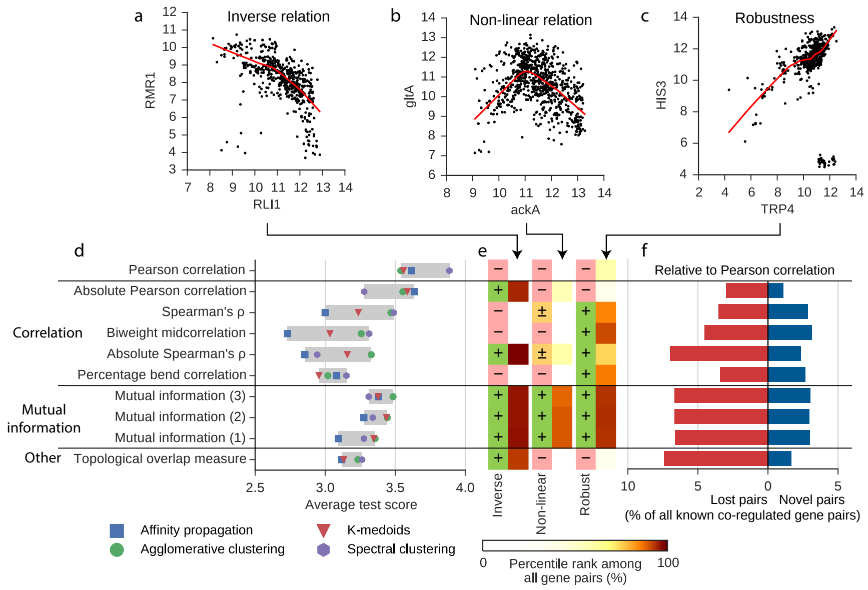


Figure 3.7: Effect of alternative similarity metrics on the performance of clustering methods. One of the most important parameters for some clustering methods is the similarity or distance measure used to compare genes. The most popular measure, the Pearson correlation, assesses the extent towards which the expression of two genes is linearly correlated among all samples. Several alternative measures have been proposed, for handling inverse relationships, non-linear effects or improve the robustness of the measure. Here we evaluated these alternative measures on four of the top clustering methods which require a similarity or distance matrix as input. **a** Example of an inverse relation between two known co-regulated genes (RLI1 and RMR1) in the DREAM5 yeast dataset. **b** Example of a non-linear relation between two known coregulated genes (gltA and ackA) in the DREAM5 *E. coli* dataset. **c** Example of a relation between two known co-regulated genes (TRP4 and HIS3) with a skewed distribution and outliers. **d** Performance of four clustering methods with different similarity measures, averaged over datasets and module definitions. **e** For every limitation of the Pearson correlation we assessed whether alternative measures can handle it theoretically (+,± and -). Can the metric handle inverse relations (+)? Can the metric detect non-linear monotonic relations (±) or more complex non-linear relations (+)? Can the method either handle outliers and/or skewed distributions (+)? Shown next to the theoretical properties are three case studies from a, b and c. Given are the rank percentages of every case study among all gene pairs in the datasets (higher is better). **f** Percentage of known co-regulated gene pairs removed (red) and gained (blue) between the Pearson correlation and an alternative metric within the top 10% of all gene pairs.

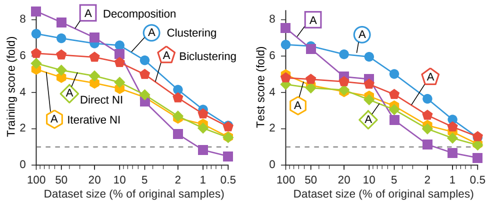


Figure 3.8: Influence of the number of samples on the performance of the top module detection methods. Shown are average training scores (left) and test scores (right) over all datasets and module definitions at different levels of random subsampling (five repeats)

We also analysed the noise sensitivity of the different methods by applying different levels of noise on the synthetic datasets. Although we saw that most methods were similarly sensitive to noise compared with their overall performance, we found that some methods, notably WGCNA and fuzzy c-means, were more sensitive (Supplementary Fig. 14).

Discussion

Unsupervised data analysis has the potential to provide an unbiased and global overview of biological datasets. Compared with other unsupervised clustering tasks in biology (extensively evaluated elsewhere [26]), module detection in gene expression data is unique, because the complexity of the underlying gene regulatory network poses particular challenges, such as local co-expression and overlap. These challenges led to the development of numerous algorithms and tools specifically dedicated to gene expression data; however, so far the comparative performance of these methods was unclear. In this work we therefore introduced a general framework for evaluating module detection methods and used it to provide a first comprehensive evaluation of state-of-the-art module detection methods for gene expression data. Based on this evaluation we analysed several practical aspects of module detection, such as the choice of methods and parameter estimation, which are summarised in Figure 3.9 and will be further discussed here. Moreover, we also provide several guidelines for further development of these methods combined with what in our view has already been accomplished, as summarised in Figure 3.10.

Module detection in gene expression data can serve a variety of roles and different methods are better suited for particular roles (Figure 3.9a,b). Owing to the ease of visualisation and interpretation, non-overlapping clustering methods can quickly generate a global overview of the dataset, revealing the main expression and functional effects among the different biological samples in the dataset [2]. Our analysis

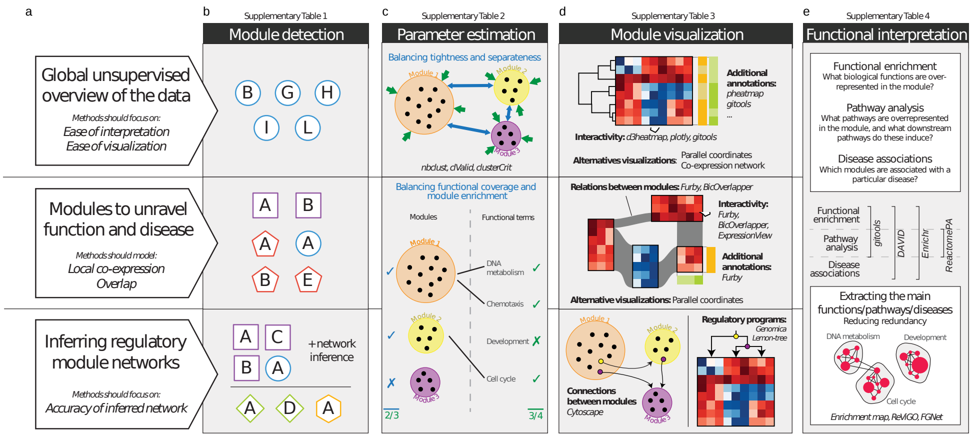

a
b
Supplementary Table 1
Module detection
c
Supplementary Table 2
Parameter estimation
d
Supplementary Table 3
Module visualization
e
Supplementary Table 4
Functional interpretation
Global unsupervised overview of the data
Methods should focus on:
Ease of interpretation
Ease of visualization
Modules to unravel function and disease
Methods should model:
Local co-expression
Overlap
Inferring regulatory module networks
Methods should focus on:
Accuracy of inferred network
+network inference
Balancing tightness and separateness
nbclust, dValid, clusterCrit
Balancing functional coverage and module enrichment
Modules
Functional terms
DNA metabolism
Chemotaxis
Development
Cell cycle
2/3
3/4
Additional annotations:
pheatmap
gitools
Interactivity: d3heatmap, plotly, gitools
Alternatives visualizations:
Parallel coordinates
Co-expression network
Relations between modules: Furby, BiCOverlapper
Interactivity:
Furby, BiCOverlapper, ExpressionView
Additional annotations:
Furby
Alternative visualizations: Parallel coordinates
Regulatory programs:
Genomica
Lemon-tree
Connections between modules
Cytoscape
Functional enrichment
What biological functions are over-represented in the module?
Pathway analysis
What pathways are overrepresented in the module, and what downstream pathways do these induce?
Disease associations
Which modules are associated with a particular disease?
gitools
DAVID
Enrichr
ReactomePA
Extracting the main functions/pathways/diseases
Reducing redundancy
Enrichment map, ReViGO, FGNet

Figure 3.9: Practical guidelines for module detection in gene expression data. Module detection in gene expression data has three main applications (left; panel **a**). For each application, we suggest different module detection methods (**b**), which in turn influences the way parameters are estimated (**c**), how the modules can be visualised (**d**), and how they can be functionally interpreted (**e**)

showed that FLAME, WGCNA, Affinity Propagation, Markov clustering (MCL), and Spectral clustering are particularly suited for such an analysis, outperforming other clustering methods on most datasets. However, because clustering methods do not detect local co-expression effects, they could potentially miss relevant modules or exclude important genes from a module. In use cases where it can be desirable that all modules are discovered in a dataset, e.g., to generate signatures for disease, therapy and prevention [4, 11, 32], or to find a set of genes responsible for a biological function, methods that detect such local co-expression and/or overlapping modules could therefore provide a substantial advantage. Consistent with this, we found that decomposition methods based on ICA were better at recovering known modules consistently across datasets. Although a handful of studies have already shown the potential of these methods in true biological settings [16, 23, 32], this was never shown in direct comparison to alternative methods and/or based on objective benchmarks. Finally, a third major application of co-expression modules is in the inference of gene regulatory networks, where modules can be used to improve the network by combining information from several genes [33] but can also improve the ease of interpretation. When we assessed the accuracy of the inferred network by combining a state-of-the-art network inference methods with different module detection methods, we found that ICA-based decomposition methods lead to the highest improvement in accuracy (primarily on yeast and synthetic datasets), closely followed by clustering and graph clustering methods (Supplementary Fig. 16). For most methods, free implementations are available either with a graphical or programming interface, of which we give an overview in Supplementary Table 1.

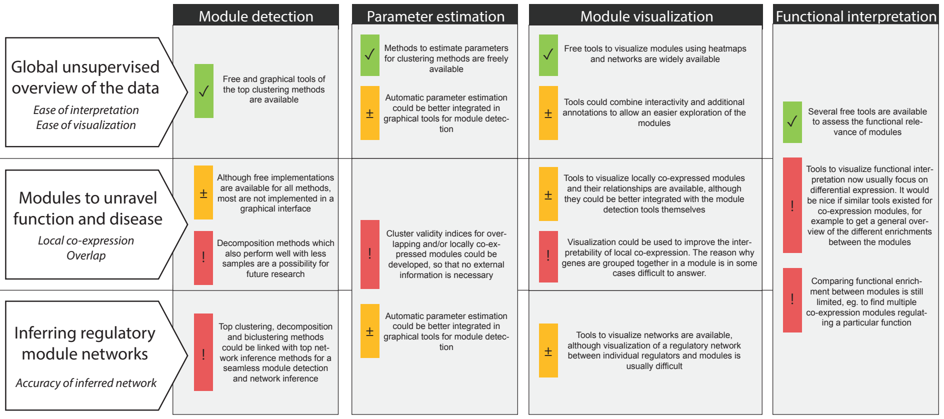
The choice of method influences subsequent steps of parameter estimation, visualisation, and functional interpretation (Figure 3.9c–e). For parameter estimation we found that cluster validity indices, the Davis–Bouldin and Kim–Ramakrishna indices in particular, are sufficient to estimate the parameters for most top clustering methods. However, the performance of these measures on alternative module detection methods was generally worse than randomly selecting parameters. For these methods, biclustering, decomposition, and direct NI in particular, we found that a measure based on functional enrichment provides a better alternative (Figure 3.9c). The kind of visualisation of the modules also heavily depends on the method. Although the results of a non-overlapping clustering analysis can be readily visualised using heat maps or networks [34], visualising overlapping modules requires more complex and hierarchical visualisations (which, e.g., indicate the overlap between modules) and is still an active research field [35, 36] (Figure 3.9d). In both cases, additional annotations can be added to the visualisation to improve interpretation of the modules, e.g., to indicate the functional annotation of the genes, and interactivity can be used to accelerate the exploration of the modules. Finally, several tools and databases can be used to functionally interpret the modules, to analyse what biological functions and pathways are enriched within the modules or to find whether the module could be associated with certain diseases. To reduce redundancy in the results of such enrichment analysis, alternative visualisation and trimming methods can be used to extract the main biological functions, pathways, or diseases associated with particular modules (Figure 3.9e). We give an overview of freely available methods that can be used to interpret co-expression modules in Supplementary Tables 2–5.

Apart from those listed in Figure 3.3b, there are also several other characteristics of module detection methods, which are important to consider in practice. Non-exhaustive module detection methods, which include some clustering methods such as FLAME and WGCNA, do not necessarily assign every gene to at least one module. Although this has the advantage that the method itself detects noisy expression profiles, users should be aware that it can also remove a lot of relevant expression profiles if the parameter values are too stringent. Network inference-based methods are unique among the different approaches, because they also generate hypotheses that can explain molecularly why certain genes are grouped in a module. Despite their lower performance according to our evaluation, they could therefore still be advantageous in certain use cases. Finally, we note that some methods are stochastic, and to assure the robustness of the results users should consider re-running the methods several times in different random states. We list these different properties of a method in Supplementary Note 2, along with a brief discussion about their algorithmic approach, important parameters, and directions to freely available implementations.

As most of the evaluation studies in the past focused only on a limited number of methods, a direct comparison with our evaluation study is difficult [17, 19, 20, 21, 22]. Furthermore, in these evaluations major conclusions frequently rely on synthetic datasets, and although it can certainly give insights into assumptions made by certain algorithms, it cannot be used to make conclusions about the usefulness of a particular algorithm on real datasets. Still, we acknowledge two noteworthy differences as follows: most studies find that biclustering methods outperform clustering [19, 20] or observe substantial performance differences between graph-based, hierarchical, and representative-based clustering methods [21, 22]. We relate these differences mainly to issues with parameter estimation, reliance on synthetic datasets and use of a limited number of evaluation metrics. In Supplementary Note 4 we give an overview of past evaluation studies, the methods they evaluated, and the evaluation aspects where we believe these studies are lacking. Similar to a recent study evaluating clustering methods on several biological datasets [26], we found that no single clustering is the best performer on all datasets, although certain methods are certainly better than others at retrieving the known modular structure within the data.

Nonetheless, we acknowledge that our evaluation workflow still has some limitations for particular applications. As we wanted to make sure that most of the modules present in our gold standard were also differentially expressed in the expression data, we used large expression compendia from very different biological conditions. However, this means that when expression differences are very subtle, other methods such as biclustering could perform better. Indeed, some biclustering methods such as FABIA are frequently used in drug discovery[37]. An evaluation focusing on these kind of subproblems is still a possibility for future research.

The detection of overlapping and locally co-expressed modules has been a longstanding challenge in transcriptomics research. Despite great efforts towards the development of these methods, their application on real biological data has been hampered because of several practical challenges. Foremost, the visualisation and interpretation of overlapping and locally co-expressed modules is more difficult. Despite some efforts [35, 36], current visualisation tools, e.g., do not directly show why certain genes are grouped in a module, which can make the module detection methods seem like a black box with unclear biological relevance. Moreover, decomposition and biclustering methods usually have several parameters, which need to be tuned on a dataset and which can affect the biological interpretation. Although we found that external functional information can be used to estimate the parameters of these methods on most datasets, the requirement for such external information can limit their applicability on well-studied organisms. Parameter estimation of biclustering and decomposition methods, which uses only the expression matrix itself, therefore remains an open issue. Finally, our evaluation also indicates that the top performing decomposition methods are much more sensitive to the number of samples in a dataset and are outperformed by clustering methods when the number of samples is limited. We anticipate that improvements on these points (visualisation, parameter estimation, and data requirements), will allow these advanced module detection methods to gain more traction in biological research. We list some past accomplishments and points for future research in Figure 3.10.



	Module detection	Parameter estimation	Module visualization	Functional interpretation
Global unsupervised overview of the data *Ease of interpretation* *Ease of visualization*	✓ Free and graphical tools of the top clustering methods are available	✓ Methods to estimate parameters for clustering methods are freely available ± Automatic parameter estimation could be better integrated in graphical tools for module detection	✓ Free tools to visualize modules using heatmaps and networks are widely available ± Tools could combine interactivity and additional annotations to allow an easier exploration of the modules	✓ Several free tools are available to assess the functional relevance of modules
Modules to unravel function and disease *Local co-expression* *Overlap*	± Although free implementations are available for all methods, most are not implemented in a graphical interface ! Decomposition methods which also perform well with less samples are a possibility for future research	! Cluster validity indices for overlapping and/or locally co-expressed modules could be developed, so that no external information is necessary	± Tools to visualize locally co-expressed modules and their relationships are available, although they could be better integrated with the module detection tools themselves ! Visualization could be used to improve the interpretability of local co-expression. The reason why genes are grouped together in a module is in some cases difficult to answer.	! Tools to visualize functional interpretation now usually focus on differential expression. It would be nice if similar tools existed for co-expression modules, for example to get a general overview of the different enrichments between the modules
Inferring regulatory module networks *Accuracy of inferred network*	! Top clustering, decomposition and biclustering methods could be linked with top network inference methods for a seamless module detection and network inference	± Automatic parameter estimation could be better integrated in graphical tools for module detection	± Tools to visualize networks are available, although visualization of a regulatory network between individual regulators and modules is usually difficult	! Comparing functional enrichment between modules is still limited, eg. to find multiple co-expression modules regulating a particular function

Figure 3.10: Recommendations for future development for the detection and interpretation of modules in gene expression data.
Counterpart of Figure 4.8 for developers of module detection methods. We list some aspects for developers of methods which have already been accomplished (green), primarily with regards to generating a global unsupervised overview of the data using clustering methods. In addition, we list some ongoing challenges, primarily with the parameter estimation, visualisation and interpretation of biclustering and decomposition methods (orange and red).

Methods

Regulatory networks and module definitions

For *E. coli* datasets, we used a regulatory network from the RegulonDB database version 8 (regulondb.ccg.unam.mx, accessed 03/06/2015), a database integrating both small-scale experimental evidence as well as genome-wide data of transcriptional regulation [38]. We only included interactions with at least one strong evidence type (APPH, BPP, FP, IDA, SM, TA, CHIP-SV, GEA, ROMA, and gSELEX). We did not group the regulatory interactions at operon level, as we found that this has only minimal impact on the overall ranking of the different methods (Supplementary Fig. 17a). We also did not include sigma factor regulations as we found that this would have a negligible effect on performance (Supplementary Fig. 17b). For the yeast datasets we used two regulatory networks. One network was generated from an integration of chromatin immunopurification-on-chip data and conserved binding motifs as described by MacIsaac et al. [39]. Another regulatory network was generated by combining genome-wide transcription factor binding data, knockout expression data, and sequence conservation [40]. We used the most stringent dataset, which required evolutionary conservation in at least two species. For the human datasets we used the 'regulatory circuits' generated by Marbach et al. [41] in which regulators were linked with target genes through a series of steps starting from binding motifs in active enhancers using FANTOM5 project data.

For every gold standard, we obtained sets of known modules based on three different module definitions. We defined minimally co-regulated modules as overlapping groups of genes that shared at least one regulator. Strictly co-regulated modules were defined as groups of genes known to be regulated by exactly the same set of regulators. Strongly interconnected known modules, on the other hand, were defined as groups of genes that are strongly interconnected, and this does therefore not necessarily reflect co-regulation. We used three different graph cluster algorithms (markov clustering, transitivity clustering, and affinity propagation) with in every case three different parameter settings representing different levels of cluster compactness. For the Markov Clustering Algorithm [42] we used inflation parameters 2, 10, and 50. For transitivity clustering [43] we used two different cutoff parameters for the fuzzy membership 0.1 and 0.9. These two parameter settings allowed the modules to overlap (Supplementary Fig. 18). In the third parameter setting for transitivity clustering, we assigned every gene to the module with the highest fuzzy membership value. For affinity propagation[44] we varied the preference value between 0.5, 2, and an automatically estimated value (see Supplementary Note 2). All known modules were then filtered for the genes present in the expression matrix. Finally, we filtered strongly overlapping known modules by merging two modules if they overlapped strongly (Jaccard coefficient > 0.8) and removed small modules by requiring at least five genes. The latter cutoff was defined based on where the average optimal performance of all methods reached a maximum.

To further validate the known modules, we assessed the extent to which the modules are co-expressed in our expression datasets. We found that all three main module definitions generate modules which are both more globally and more locally (according to extreme expression biclustering definition, see Supplementary Note 2) co-expressed compared with permuted modules (Supplementary Fig. 19). Certain module definitions, strict coregulation in particular, and datasets, *E. coli*, and synthetic data generate modules that are better co-expressed within the expression data, which could explain why module detection methods generally also perform better on these datasets and module definitions (Figure 3.3c,d). We further confirmed the biological relevance of the known modules by investigating their functional enrichment. We found that on the *E. coli* datasets, 50–70% of all functional terms (both for Gene Ontology (GO) [45] and Kyoto Encyclopedia of Genes and Genomes (KEGG) pathways [46]) were enriched in at least one known module, and that 60–80% of all known modules were enriched in at least one functional term (Supplementary Fig. 20). The coverage of the whole functional space was much less on the yeast data, with about 5–15% GO terms and 10–30% KEGG pathways covered (Supplementary Fig. 20a). On the other hand, a substantial number of all known modules were enriched in at least one functional term, ranging from 30% to 60% on GO terms and 30% to 50% on KEGG pathways (Supplementary Fig. 20b). Compared with known modules, observed modules covered the functional space in most cases a little bit better for the top methods (Supplementary Fig. 21).

Gene expression data

We used a total of nine expression datasets for the study, two from *E. coli*, two from *Saccharomyces cerevisae*, three human datasets, and two synthetic datasets. Datasets consisted of hundreds of samples in various genomic and/or environmental perturbational settings.

We obtained a first *E. coli* dataset from the Colombos database (version 2.0, www.colombos.net) [47]. This dataset is unique among the four because it does not contain raw expression values from one sample but instead contains log ratios between test and reference conditions, which allowed the authors to integrate across different microarray platforms and RNA-sequencing experiments. A second *E. coli* dataset was downloaded from the DREAM5 network inference challenge [15] website (https://www.synapse.org/#!Synapse:syn2787209/wiki/70349).

For S. cerevisiae, we aggregated an expression compendium by integrating data from 127 experiments (filtered on S. cerevisae samples) using the GPL2529 platform from Gene Expression Omnibus (https://ncbi.nlm.nih.gov/geo). Raw expression data were normalised using Robust Multichip Average as implemented in the Bioconductor affy package. A second yeast dataset was obtained from the DREAM5 website (https://www.synapse.org/#!Synapse:syn2787209/wiki/70349).

We obtained the human TCGA datasets from a pan-cancer study of 12 cancer types (https://www.synapse.org/#!Synapse:syn1715755) [48]. The human GTEX dataset, which contains expression profiles from different organs from hundreds of donors [49], was downloaded from the GTEX website (https://www.gtexportal.org). The SEEK GPL5175 dataset is an aggregation of public datasets using the GPL5175 microarray platform and was retrieved from https://seek.princeton.edu.

We generated two synthetic datasets starting from the *E. coli* regulondb network and yeast MacIsaac network (both described above) using GeneNetWeaver. This network simulator models the gene regulation using a detailed thermodynamic model and simulates this model using ordinary differential equations [50]. Different experimental conditions were simulated using the 'Multifactorial Perturbations' setting, where transcription rates for a subset of genes are randomly perturbed.

For all expression datasets we filtered out the least variable genes by requiring a minimal standard deviation in expression of 0.5 (for yeast and *E. coli*) and 1 (for human datasets). Heatmaps for every dataset can be found at Supplementary Fig. 21.

Each dataset has its own advantages and disadvantages. Real datasets better fit the real use case and are thus the most biologically relevant, although limited availability of gold standard can make an evaluation on real data challenging. Although our knowledge of the regulatory networks of model micro-organisms, primarily *E. coli*, is already substantial, it is still far from complete [51]. While evaluating on data with more complex regulatory networks such as humans is certainly necessary to ensure the broad relevance of the evaluation, the definition of gold standards on these datasets can be even more problematic because of the broad prevalence of false-positive and false-negative interactions due to a variety of reasons, such as cellular context [12] and non-functional binding [52]. We therefore also included synthetic datasets where the known regulatory network is completely given and thus estimates of both sensitivity and precision of a method can be accurately estimated. Together, we believe these datasets give complementary support to our evaluation strategy and assure its broad relevance.

Similar to a previous evaluation study of biclustering methods [53], our datasets can contain both large differences between samples, as well as small differences, as indicated by the distribution of all log-fold changes between samples (Supplementary Fig. 22).

Module detection methods

We chose a total of 42 module detection methods based on (i) a freely available implementation, (ii) performance within previous evaluation studies [17, 19, 20, 21], and (iii) novelty of the algorithm. See Supplementary Note 2 for a brief overview of every method and Supplementary Table 1 for an overview of the implementations used in this study and alternative implementations. We classified all module detection methods in five major categories. We acknowledge however that the boundaries between the different categories are not always clear, as certain clustering and biclustering methods, e.g., also use a matrix decomposition step within their algorithm. The common theme of clustering methods is that they group genes according to a global similarity in gene expression. Even if clustering methods can detect (after some post-processing) overlapping clusters, this overlap is detected only because a certain gene is still globally similar to both two clusters, and not necessarily because of a local co-expression. Decomposition methods try to approximate the expression matrix using a product of smaller matrices. Two of these matrices contain the individual contributions of respectively genes and samples to a particular module. As samples are allowed to contribute to a particular module only to a certain degree, decomposition methods can detect local co-expression. Related to these methods are biclustering methods, which detect groups of genes, and samples, which show some local co-expression only within the bicluster. In biclustering, samples either contribute to a particular module or not, in contrast to decomposition methods where all samples contribute to a certain extent. Modules detected by biclustering methods can therefore be easier to interpret compared with those of decomposition methods, as the exact origin of the local co-expression is better defined. In some cases, a biclustering method is simply an extension of an existing decomposition method but with an extra requirement that the contribution of a gene and sample to a module is sparse (i.e., contains lots of zeros). Direct NI methods try to generate a simple model of gene regulation, in most cases by using the expression matrix to assign a score to every regulator-gene pair [15]. Although their primary application is to predict novel regulatory relationships between genes, some studies have also used the resulting weighed regulatory network to detect gene modules [54, 55]. A list of regulators was generated for *E. coli* by looking for genes annotated by GO with either "transcription, DNA-templated," or "DNA binding," and for yeast and human with "sequence-specific DNA-binding RNA polymerase II transcription factor activity." The same list was also used for iterative NI methods, which start from an initial clustering, and iteratively refine this clustering and an inferred regulatory network.

We further classified clustering methods according to their "induction principle," a classification that does not use the way clusters are represented in the algorithm (the model), but rather looks at the optimization problem underlying the clustering algorithm [56]. Graph-based clustering algorithms make use of graph-like structures, such as K-nearest-neighbour graphs and affinity graphs, and group genes if they are strongly connected within this graph-like structure. Representation-based methods iteratively refine a cluster assignment and representative (such as the centroid) of the cluster. Hierarchical clustering methods construct a hierarchy of all the genes within the expression matrix. Finally, density-based methods detect modules by looking at contiguous regions of high density. It should also be noted that some clustering methods use elements from multiple categories. FLAME (clustering method A), e.g., uses elements from graph-based, representative-based, and density-based clustering, whereas affinity propagation contains both elements from graph-based and representative-based methods. In cases like this, we ultimately classified an algorithm based on which aspect of the algorithm we believe has the major impact on the final clustering result. Biclustering methods were further classified according to the type of biclusters they detect. The expression within constant biclusters remains relatively stable, whereas the genes within extreme biclusters have a relatively high expression in a subset of conditions compared with other genes. The expression within pattern-based biclusters follow more complex models such as additive models [57], multiplicative models [53], and coherent evolution [58].

Post-processing steps were required for certain methods to get the results in a correct format for comparison with the known modules. All parameters for these post-processing steps were optimised within the grid search approach (as described in Supplementary Note 2 for every method). For fuzzy clustering methods, we obtained crisp but potentially overlapping modules by placing a cutoff on the membership values. For direct NI methods, we first used a cutoff to convert the weighed to an unweighted network, and then detected modules using the same module definitions as previously described. For decomposition methods we explored several post-processing steps in literature (see Supplementary Note 2).

As gene regulatory networks, even in these model organisms, are still very incomplete [51], a small majority of the genes was not included in any known module (Supplementary Fig. 23). Although we did retain these genes in the expression matrix before module detection, we removed these genes in the observed modules before scoring. This was to avoid a strong overestimation of the number of false positives in the observed modules, as most of these genes probably belong to one or more co-regulated modules, which we do not yet know. Finally, similar to the known modules, we filtered the observed modules so that each module contained at least five genes.

Similar to our analysis with known modules, we assessed the extent to which the genes detected by each of the methods are co-expressed in the datasets based on three co-expression metrics inspired by the three types of biclustering metrics (Supplementary Fig. 24). (1) An overall co-expression metric using the average correlation, (2) an extreme expression metric by looking at the top 5% average z-scores for every gene in the module, and (3) the root mean-squared deviation within the expression values of each module. For each metric, we compared the distribution of the real modules with permuted modules by calculating the median difference using the wilcox.test function in R. We found that every module detection method found modules which were more strongly co-expressed than permuted modules. Compared with the co-expression of known modules, the module detection methods also produced modules that are more strongly co-expressed. Specifically for biclustering methods, we also investigated the co-expression only in those samples within each bicluster. Here we found that, except for some pattern-based biclustering methods, most biclustering methods detected the type of modules, which they are designed to detect (Supplementary Fig. 24).

Parameter tuning

Parameter tuning is a necessary but often overlooked challenge with module detection methods. All too often, evaluation studies use default parameters which were optimized for some specific test cases by the authors. This does not correspond well with the true biological setting, where some parameter optimization is almost always necessary to make sense of the data. Therefore, to make sure an evaluation is as unbiased as possible, some parameter optimization is always required. However, one should be careful of overfitting parameters on specific characteristics of one dataset, as such parameters will lead to suboptimal results when generalizing the parameter settings to other datasets. This could again introduce a bias in the analysis, where methods with a lot of parameters would better adapt on particular datasets, but would not generalise well on other datasets. In this study we tried to address both problems as follows. We first used a grid search to explore the parameter space of every method and determine their most optimal parameters given a certain dataset and module definition, which resulted in a training scores. Next, in a process akin to cross-validation, we used the optimal parameters of one training dataset from another organism to score the performance on another test dataset, which resulted in a test scores for every training dataset. As we saw that optimal parameters were in most cases very different between synthetic and real datasets, we only used real datasets to train parameters for other real datasets and synthetic datasets for other synthetic datasets. We refer to Supplementary Note 2 for an overview of what parameters were varied for every method.

Evaluation metrics

We used four different scores to compare a set of known modules with a set of observed modules and, after normalisation, combined them in one overall score. Note that classical scores comparing clusterings, such as the Rand index, the F1, or the normalised mutual information, could not be used as these scores are unable to handle overlap and/or overlap [59] (Supplementary Note 1). The recall and specificity within the recently proposed CICE-BCubed scoring system measure whether the number of modules containing a certain pair of genes is comparable between the known and observed modules[60]. It is based on the Extended BCubed [59], but reaches the perfect score of 1 only when both known and observed overlapping clusterings are equal. If G represents all genes, M a set of known modules, M' a set of observed modules, M(g) the modules that contain g, and E(g, M) the set of genes that are together with g in at least one module of M (including g itself), the precision is defined as:

$$\text{Precision} = \frac{1}{|G|}\sum_{g\in G}\frac{1}{|E(g,M')|}\sum_{g'\in E(g,M')}\frac{\min(|M'(g)\cap M'(g')|,|M(g)\cap M(g')|)\cdot\Phi(g,g')}{|M'(g)\cap M'(g')|}$$

where

$$\Phi(g,g') = \frac{1}{|M'(g,g')|}\sum_{m'\in M'(g,g')}\max_{m\in M(g,g')}\text{Jaccard}(m',m)$$

Recall is calculated in the same way but with M' and M switched. The recovery and relevance scores, which have been previously used in evaluation studies of biclustering methods, assess whether each observed module can be matched with a known module and vice versa [17, 19]. Relevance is defined as

$$\text{Relevance} = \frac{1}{|M'|}\sum_{m'\in M'}\max_{m\in M}\text{Jaccard}(m',m)$$

Recovery is calculated in the same way but with M' and M switched.

Before combining scores across different datasets and module definitions, we first normalised every score by dividing it by an average score of 500 permuted versions of the known modules (Supplementary Fig. 25). The goal of this step was to prevent easier module structures (small modules, low number of modules, and no overlap) of certain module definitions and datasets from disproportionally influencing the final score. Permuted modules were generated by randomly mapping the genes of a dataset to a random permutation of the genesand replacing every occurrence of a gene in a known module with its mapped version. Based on this random model, module structure (size, number, and overlap) remained the same, while only the assignment changed. We also tested two alternative random models. The STICKY random model has been previously described [61]. We adapted this model for directed networks by calculating the stickiness index separately for incoming and outgoing edges. For the scale-free network [62], we used the networkx Python package with default parameters.

We finally calculated the harmonic mean between the normalised versions of all four scores to obtain a final score representing the performance of a particular method on a given dataset and module definition.

For human data we used an alternative score that assesses the extent to which the targets of every regulator are enriched with at least one module of the dataset. As described earlier, we used the clustered version of the regulatory circuits dataset [41], which contains weights for every regulator and target gene combination across 32 tissue and cell-type contexts. For every combination of target genes and observed module we calculated a p-value of enrichment using a right-tailed Fisher's exact test (corrected for multiple testing using the Holm–Šídák procedure [63]) and the strength of this enrichment using the odds ratio. Although we calculated these values within every cell type and tissue context separately, we retained for every regulator its minimal p-value and the corresponding odds ratio across the different contexts, as we do not know the exact cell-type and tissue context in which the genes of the observed modules are co-expressed. We then extracted for every regulator its maximal odds ratio across the observed modules where the targets of the regulators were enriched (corrected p-value < 0.1). The aucodds score was then calculated by measuring the area under the curve formed by the percentage of regulators with an odds ratio equal or larger than a particular cutoff and the log10 odds ratios within the interval 1 and 1000-fold enrichment. To work in a cutoff-independent manner we averaged the aucodds scores over a range of weight cutoffs. Performance generally decreased with more stringent cutoffs (Supplementary Fig. 26a,b) although some biclustering methods and direct NI methods remained more stable across a wide range of cutoff values (Supplementary Fig. 26c,d). This score was normalised in a similar way as previously described, where the initial known modules were defined using the minimal co-regulation module definition and subsequently randomly permuted by mapping the genes within the modules to a random permutation.

We reweighted the scores between datasets and module definitions using a weighted mean so that module definitions (minimal co-regulation, strict co-regulation, and interconnected subgraphs) and each organism (*E. coli*, yeast, human, and synthetic) had equal influence on the final score.

Influence of overlap

We split the genes of every datasets based on whether they belonged to only one or multiple modules using the minimal co-regulation module definition. If G* denotes such a subset of genes in the expression matrix, we calculated a precision* score specifically for this subset using:

$$\text{Precision} = \frac{1}{|G*|}\sum_{g\in G*}\frac{1}{|E(g,M')|}\sum_{g'\in E(g,M')}\frac{\min(|M'(g)\cap M'(g')|,|M(g)\cap M(g')|)\cdot\Phi(g,g')}{|M'(g)\cap M'(g')|}$$

A Recall* score was calculated similarly but with M' and M switched. A final score for a particular set of genes was obtained by taking the harmonic mean between the normalised versions of the Recall* and Precision*.

Automatic parameter estimation

The four cluster validity indices evaluated in this study all performed well in a recent evaluation study and are defined there [28]. Most indices try to optimise the balance between tightness (the expression variability within a module) and separation (the expression differences between modules). For metrics requiring a distance matrix, we subtracted the absolute Pearson's correlation from one.

We also investigated two metrics to assess the functional coherence of the modules according to the GO database [45] and the KEGG pathways database [46]. We filtered redundant gene sets by, starting from the largest gene set, removing gene sets if they overlap too much with larger non-removed gene sets (Jaccard index > 0.7). The biological homogeneity index measures the proportion of gene pairs within a module which are also matched within a functional class [22]. For the F-aucodds score we calculated an aucodds score as described earlier in both the dimension of the gene sets (denoting how well all functional sets are covered by the observed modules) and the dimension of the observed modules (denoting how well the modules are enriched in at least one function gene set), and combined both scores by calculating its harmonic mean.

As automatic parameter estimation performed very poorly on non-exhaustive module detection methods (which include some clustering methods, see Supplementary Note 2), we assigned every unassigned gene to the module with which the average correlation was the highest prior to calculating the cluster validity indices.

Similarity measures

For clustering methods requiring a similarity matrix as input, we used the Pearson's correlation in our initial evaluation. For methods requiring a dissimilarity matrix, we subtracted the Pearson's correlation values from two. For the comparison of different similarity measures, we selected four top clustering methods that require a similarity measure as input. We compared a total of 10 alternative measures that are briefly described in Supplementary Note 3 along with directions to implementations. We did not evaluate the distance correlation, percentage bend correlation, Hoeffding's D, and maximal information coefficient [64], because they required excessive amounts of computational time and/or memory, which would be impractical for module detection in general use cases. To convert a similarity matrix to a dissimilarity matrix or vice versa, we subtracted the values from the maximal value between all gene pairs on a given dataset. To determine the influence of an alternative similarity measure on the performance of clustering methods, we re-ran all parameter optimization steps for every alternative measure and again calculated test scores as described earlier.

Code availability

Code to evaluate module detection methods and further expand the evaluation are available as Jupyter Notebooks [65, 66] (https://www.jupyter.org) at https://www.github.com/saeyslab/moduledetection-evaluation.

Supplementary Note 1: Measures for comparing overlapping modules

Numerous scores have been proposed to compare clusterings of data [67, 26, 68, 59], but most of these scores have problems with handling overlap[1] and/or non-exhaustive cluster assignment[2], which has already been discussed elsewhere [59] and which we here further illustrate using 12 small test cases (**Figure 3.11**). We define two different sets of known modules, without overlap (1-6) and with overlap (7-12). A perfect match between the observed modules and known modules is given in case 1 and 7. In every other test case the observed modules do not perfectly correspond with the known modules, and therefore the score of these test cases should become worse compared to case 1 or 7. However, as shown in **Table 3.2**, none of the classical clustering scoring metrics fulfill this criterion. Most scores have issues when the known modules and/or observed modules overlap with each other, as the performance on cases 4-6 and 11-12 stays the same or even increases compared to the perfect case. Only one score can perfectly handle overlap (F-measure [26]), but it has problems handling non-exhaustive cluster assignment, as evidenced by its perfect score on cases 2 and 9.

Several alternative scores have been proposed in literature to better handle potential overlap between clusters/modules. In the following formulas, we use these conventions: G represent all genes, M a set of known modules, M' a set of observed modules, $M(g)$ the modules which contain g and $E(g, M)$ the set of genes which are together with g in at least one module of M (including g itself).

One family of measures, which includes the recovery and relevance scores used in this study, have already been extensively applied within the biclustering literature [19, 17]. Similar scores have also independently been described elsewhere [69, 70]. These scores are calculated in two steps. First a similarity/distance matrix is calculated between the two sets of modules. There are several possibilities for this similarity score, such as the Jaccard index [19] or entropy based measures [69]. In the next step the similarity values are summarised in one number by mapping known modules to observed modules and vice versa. A score quantifying the false positives (S_1)

[1]Defined as at least one gene belonging to multiple modules
[2]Defined as certain genes not included in any modules

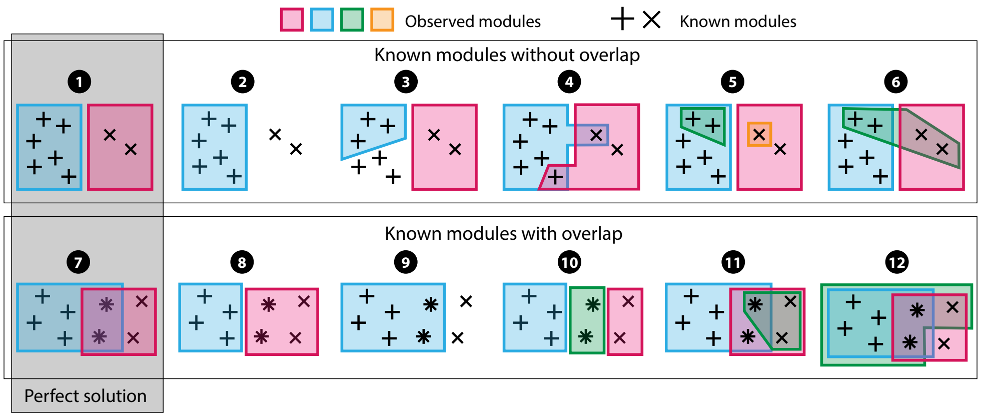


Figure 3.11: 12 test cases to assess scores for comparing two sets of potentially overlapping modules.

is calculated by summing/averaging the similarities for every observed modules by selecting the best representative in the known modules. Similarly, a score quantifying the false negatives (S_2) is calculated by summing/averaging the similarities for every known modules by selecting the best representative in the observed modules. These two scores can then be combined in a final score giving the trade-off between false positives and false negatives by summing or averaging S_1 and S_2.

[71] used an asymmetric measure for module similarity:

$$\begin{aligned} S_1 &= \text{Sensitivity} = \frac{1}{|M'|} \sum_{m' \in M'} \max_{m \in M} \frac{|m' \cap m|}{|m|} \\ S_2 &= \text{Precision} = \frac{1}{|M|} \sum_{m \in M} \max_{m' \in M'} \frac{|m' \cap m|}{|m'|} \\ S &= \frac{2}{\frac{1}{S_1} + \frac{1}{S_2}} \end{aligned} \quad \textbf{(Score 1)}$$

The ”Precision” score was originally named the ”Specificity” in this study, but the actual meaning relates more closely to the common usage of precision as it estimates how well the observed modules are also known.

[19] used a symmetric measure for module similarity, the Jaccard index:

	Known modules without overlap						Known modules with overlap					
	1	2	3	4	5	6	7	8	9	10	11	12
F-measure [67]	1.00	0.75	0.50	1.11	1.21	1.33	1.25	0.90	0.75	0.73	1.57	2.06
F-measure [26]	1.00	1.00	0.77	0.89	0.81	0.77	1.00	0.89	1.00	0.67	0.96	0.97
1-FDR [26]	1.00	0.94	0.25	1.00	1.00	1.00	1.00	0.60	0.75	0.40	1.00	1.00
1-FPR [26]	1.00	0.92	0.50	1.00	1.00	1.00	1.00	0.50	0.62	0.40	1.00	1.00
FMI [26]	1.00	0.97	0.50	0.83	1.00	0.89	1.00	0.77	0.87	0.63	1.00	0.91
Jaccard [26]	1.00	0.94	0.25	0.70	1.00	0.80	1.00	0.60	0.75	0.40	1.00	0.83
Rand [26]	1.00	0.96	0.57	0.75	1.00	0.86	1.00	0.71	0.82	0.57	1.00	0.86
Sensitivity [26]	1.00	1.00	1.00	0.70	1.00	0.80	1.00	1.00	1.00	1.00	1.00	0.83
Specificity [26]	1.00	0.92	0.50	1.00	1.00	1.00	1.00	0.50	0.62	0.40	1.00	1.00
V-measure [26]	1.00	0.00	1.00	0.22	0.67	0.49	0.13	0.43	0.00	0.46	0.12	0.07
Purity [68]	1.00	0.75	0.62	1.00	1.38	1.25	1.25	1.00	0.75	1.00	1.62	2.00
Entropy [68]	0.56	0.00	0.67	0.61	1.17	1.01	0.68	0.67	0.00	1.05	1.06	1.09

Table 3.2: Comparison of different measures for comparing two sets of modules, based on the test cases described in Figure 3.11.
These metrics are frequently used to compare different non-overlapping and exhaustive clusterings. Compared with the score on test cases 1 and 7 (grey), a good measure should consequently score lower on cases 2-6 and 8-12 (green). This condition is not satisfied by any of the measures.

$$
\begin{aligned}
S_1 = \text{Recovery} &= \frac{1}{|M|} \sum_{m \in M} \max_{m' \in M'} \text{Jaccard}(m', m) \\
S_2 = \text{Relevance} &= \frac{1}{|M'|} \sum_{m' \in M'} \max_{m \in M} \text{Jaccard}(m', m) \\
\text{Jaccard}(m', m) &= \frac{|m' \cap m|}{|m' \cup m|} \\
S &= \frac{2}{\frac{1}{S_1} + \frac{1}{S_2}}
\end{aligned}
\qquad \textbf{(Score 2)}
$$

[53] proposed a slightly modified version of the Recovery and Relevance. They added an additional constraint so that every known module can only be mapped to one observed module and vice versa. If $p_i = \{m_i, m_i'\}$ represents a pair of a known module m and observed modules m', the consensus score is defined as

$$
\text{Consensus} = \frac{1}{\max\left(|M|, |M'|\right)} \sum_{p_i \in P} \text{Jaccard}(m', m) \qquad \textbf{(Score 3)}
$$

The known modules and observed modules are matched with each other so that the consensus score is maximised using the Hungarian algorithm (**Score 3**).

[70] proposed the Best Match scores, using the edit distance, jaccard index and an entropy based measure (based on earlier work by [69]) as similarity measures.

$$\begin{aligned} S_1 =& \frac{1}{|M|} \sum_{m \in M} \max_{m' \in M'} \text{Jaccard}(m', m) \\ S_2 =& \frac{1}{|M'|} \sum_{m' \in M'} \max_{m \in M} \text{Jaccard}(m', m) \\ S =& S_1 + S_2 \end{aligned} \tag{Score 4}$$

$$\begin{aligned} S_1 =& \frac{1}{|M|} \sum_{m \in M} \max_{m' \in M'} H(m'|m) \\ S_2 =& \frac{1}{|M'|} \sum_{m' \in M'} \max_{m \in M} H(m|m') \\ S =& S_1 + S_2 \end{aligned} \tag{Score 5}$$

Although **Score 2** and **Score 4** have the same S_1 and S_2, they differ in the way these two scores are aggregated, i.e. a harmonic mean (**Score 2**) and a summation (**Score 4**). We did not consider the other scores proposed by [70] because they require one or more parameters and would add another source of potential bias in the analysis.

Another family of measures is based on the BCubed measure. First proposed in [72] to compare non-overlapping clustering, [59] extended this measure to also handle overlap. [60] adapted the metric to make sure it can only reach the optimal value of 1 when the observed modules are the same as the known modules:

$$\begin{aligned} S_1 = \text{Recall} =& \frac{1}{|G|} \sum_{g \in G} \frac{1}{|E(g, M)|} \sum_{g' \in E(g,M)} \frac{\min(|M'(g) \cap M'(g')|, |M(g) \cap M(g')|) \cdot \Phi(g, g')}{|M(g) \cap M(g')|} \\ \Phi(g, g') =& \frac{1}{|M'(g, g')|} \sum_{m' \in M'(g,g')} \max_{m \in M(g,g')} \text{Jaccard}(m', m) \end{aligned}$$

$$\begin{aligned} S_2 = \text{Precision} =& \frac{1}{|G|} \sum_{g \in G} \frac{1}{|E(g, M')|} \sum_{g' \in E(g,M')} \frac{\min(|M'(g) \cap M'(g')|, |M(g) \cap M(g')|) \cdot \Phi(g, g')}{|M'(g) \cap M'(g')|} \\ \Phi(g, g') =& \frac{1}{|M(g, g')|} \sum_{m \in M(g,g')} \max_{m' \in M'(g,g')} \text{Jaccard}(m', m) \end{aligned}$$

$$S = \frac{2}{\frac{1}{S_1} + \frac{1}{S_2}} \tag{Score 6}$$

While we also considered including ”module preservation statistics” as proposed by Langfelder and colleagues [73], we found that these measures are primarily useful to assess whether individual modules are preserved within a network, but not whether all (or most) modules present within a network are found by a particular module detection method.

The structure and size of modules detected by module detection methods can vary wildly between methods and parameter settings. For instance, some parameter settings of decomposition methods will only assign a small number of genes to any module. Other parameter settings will assign all genes multiple times to several large modules. A good score should be robust against such extreme cases as they could be produced by certain methods during the parameter optimization procedure. We tested this based on an empirical experiment where we used a set of known modules (from the *E. coli* COLOMBOS dataset using the minimal co-regulation module definition) and compared them with several extreme cases of observed modules (**Figure** 3.12):

- Two trivial clustering examples. Putting all genes together in one large module had bad performance for all scores except for Score 5. Putting all genes in their own separate module resulted in bad performance for all scores except for Score 4.
- Permutations of the known modules. A certain percentage of all genes is mapped to a permuted version of these genes, and all instances of a gene within the known modules are replaced by the mapped version. As expected, in all cases permuting the known modules had severe effects on performance.
- Effect of using only a subset of all known modules. Again, performance decreased consistently between all scores.
- Effect of randomly adding extra genes to the known modules. Score 5 responded very strongly to this relative to other perturbations.
- Effect of randomly removing genes from known modules. Again, performance decreased in all scores, although the effect was relatively weak for Score 5.
- Randomly sampling modules from the full solution set (all possible modules).

Overall we concluded that **Score 4** and **5** respond inconsistently in certain perturbational settings, while the other scores are more robust.

Finally, we tested whether these scores can better handle both overlap and non-exhaustive cluster assignment using the test cases from **Figure** 3.11. We found that only **Score 1** still had problems regarding overlap in a subset of cases. The other scores (**Score** 2, **3** and **6**) all performed well according to our test cases. Together with the strong theoretical background of Score 6 [59] and several examples of studies where Score 2 has been successfully applied to compare biclustering methods [19, 74, 17], we chose **Score 2** and **Score 6** for the main evaluation study. The score on the other metrics, together with the three most popular classical clustering evaluation measures are given in Supplementary Figure 8.

	Known modules without overlap						Known modules with overlap					
	1	2	3	4	5	6	7	8	9	10	11	12
Sensitivity (Score 1a) [71]	1.00	0.50	0.75	1.00	1.00	1.00	1.00	0.83	0.75	0.58	1.00	1.00
Specificity (Score 1b) [71]	1.00	1.00	1.00	0.76	1.00	0.83	1.00	1.00	1.00	1.00	1.00	0.95
F-measure (Score 1) [71]	1.00	0.67	0.86	0.86	1.00	0.91	1.00	0.91	0.86	0.74	1.00	0.98
Recovery (Score 2a) [19]	1.00	0.50	0.75	0.76	1.00	1.00	1.00	0.83	0.62	0.58	1.00	1.00
Relevance (Score 2b) [19]	1.00	1.00	0.75	0.76	0.71	0.83	1.00	0.83	1.00	0.56	0.92	0.95
F-measure (Score 2) [19]	1.00	0.67	0.75	0.76	0.83	0.91	1.00	0.83	0.77	0.57	0.96	0.98
Consensus (Score 3) [53]	1.00	0.50	0.75	0.76	0.50	0.67	1.00	0.83	0.50	0.39	0.67	0.67
Recall (Score 6a) [60]	1.00	0.75	0.34	0.81	1.00	1.00	1.00	0.56	0.64	0.30	1.00	1.00
Precision (Score 6b) [60]	1.00	0.75	0.44	0.60	0.91	0.70	1.00	0.83	0.75	0.58	0.87	0.50
F-measure (Score 6) [60]	1.00	0.75	0.39	0.69	0.95	0.82	1.00	0.67	0.69	0.40	0.93	0.67
F-measure (Score 2 and 6)	1.00	0.71	0.51	0.73	0.89	0.86	1.00	0.74	0.73	0.47	0.94	0.80

Table 3.3: Comparison of different measures for comparing two sets of modules, based on the test cases described in Figure 3.11.
Unlike the metrics in **Table** 3.2, all metrics have been developed for overlapping and non-exhaustive sets of modules. Compared with the score on test cases 1 and 7 (grey), a good score should consistently score lower on cases 2-6 and 8-12 (green).

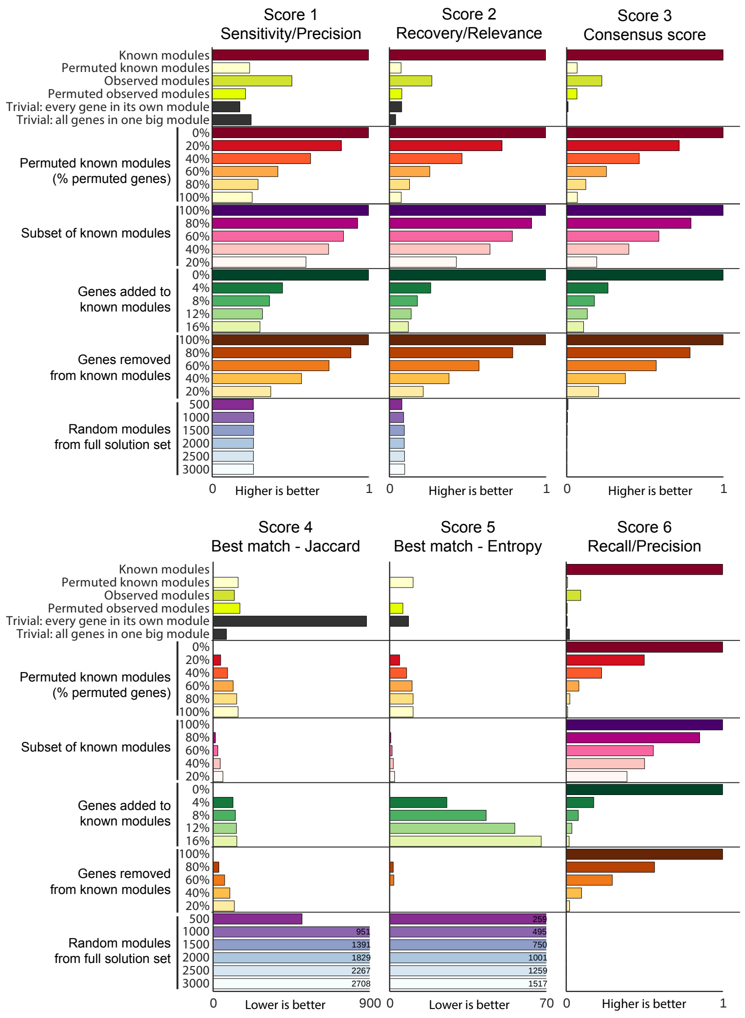


Figure 3.12: Empirical study of the robustness of several scores comparing overlapping clusters (as defined in Supplementary Note 1) in perturbational settings.
In every case, known modules (from the *E. coli* COLOMBOS dataset using the minimal co-regulation module definition) were compared to a different set of modules given in the y-axis, usually derived from the known modules but with a subset of genes permuted, a subset of modules selected or some random genes added or removed from the modules. As a reference we also give the performance of the modules detected by affinity propagation (clustering methods I) at optimal parameter settings.

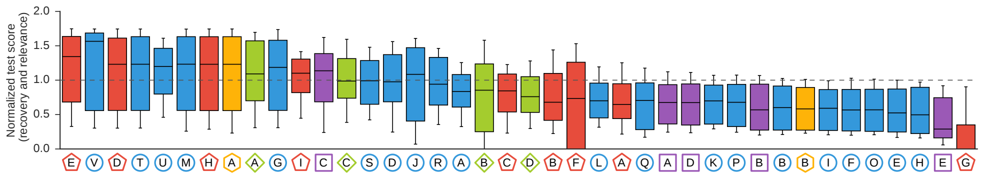


Figure 3.13: Comparing known and observed modules on human datasets.
Shown are the distribution of normalised test scores when comparing known modules with observed modules on three human datasets (GTEX, TCGA and SEEK GPL) using the Recovery and Relevance scores. Known modules were extracted from the regulatory circuits networks [41] at different cutoffs using the minimal coregulation definition (as described in the **Methods**). We found that none of the module detection methods consistently outperformed permuted known modules across the different datasets and cutoffs.

While almost all module detection methods performed better than permutations of the known modules on *E. coli*, yeast and synthetic data, we found that the performance was generally very low on human datasets, rarely reaching the performance levels of permuted modules (**Figure** 3.13). We reasoned that this was mainly because of the extremely high number of false positive interactions in current large-scale human regulatory networks due to (i) promiscuous binding, (ii) context specific regulation, (iii) the difficulty of linking binding events to the activity of a promoter and (iv) the degeneracy of binding specificity.

We therefore developed a new score (aucodds) which, instead of looking at the exact overlap between known and observed modules, will use the enrichment of known targets of a particular transcription factor within the observed modules. To calculate the aucodds score given a regulatory network and observed modules, first the enrichment of target genes is calculated for every observed module and transcriptional regulator using a Fisher's exact test. Next, after correction for multiple testing, we calculate for every regulator the best odds ratio in the modules where the regulator's target genes are enriched (q-value < 0.1). Finally, for a range of odds-ratio cutoff values the percentage of regulators with an equal or larger odds-ratio are calculated and these values are combined within a final score by calculating the area under the curve formed by the log10-cutoff values and the percentage of enriched regulators. The score therefore not only looks at whether the targets of a regulator are enriched in any of the modules, but also how strongly they are enriched.

We found the aucodds score to be more stable when false-positive interactions are added to the gold standard (**Figure** 3.14**a**), while Scores 2 and 6 quickly converged to the levels of permuted modules. Although this score is therefore much more robust against large number of false positive regulatory interactions, it conversely also makes the score less sensitive to false positive genes in the observed modules compared with previously described measures (**Figure** 3.14**b**). Nonetheless, we found the aucodds score to be highly correlated with other scores for overlapping modules on the *E. coli*, yeast and synthetic datasets across parameter settings and methods (**Figure** 3.14**c**).

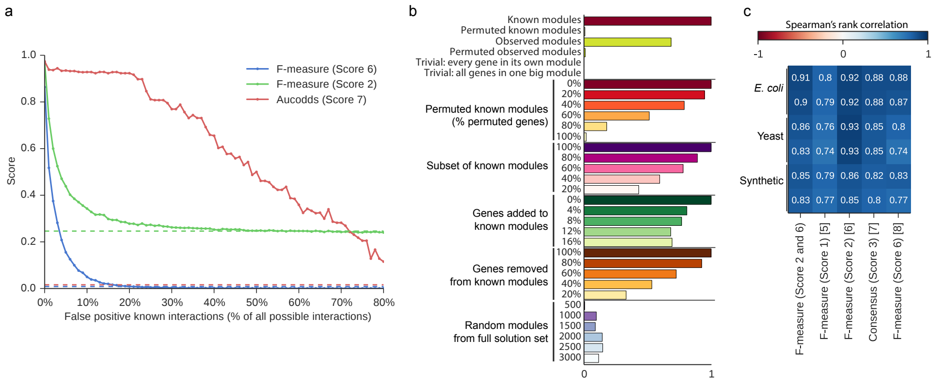


Figure 3.14: (a) The aucodds score (Score 7) decreases more slowly than other scores with increasing number of false positive regulatory interactions. **(b)** Empirical study of the robustness of aucodds score (Score 7) (as defined in **Supplementary Note 1**) in perturbational settings. See **Figure** 3.12 **(c)** Spearman correlation between the aucodds score and other scores on all perturbational settings in **(b)**.

.

Update

Since this paper was published in 2018, several other researchers have contacted me with questions on how to apply the evaluation to their own methods. This includes new (bi)clustering methods, but also alternative similarity measures or parameter estimation methods. Of note here is the QUBIC2 paper [75], that uses our benchmarking workflow to compare against alternative biclustering algorithms. The QUBIC2 method has some modifications to work on single-cell data. In hindsight, I could have focused much more on making the benchmark easier to extend, for example by using conda environments and some refactoring of my code. This especially in light of the other benchmarking study that I performed (Chapter 4).

Even though this study included only bulk transcriptomics datasets (and also still a lot of microarray data), its results might still be highly relevant for single-cell data. The current interest in the field mainly centers around finding and annotating (novel) cell populations, as exemplified by the numerous cell atlas paper [76, 77]. Even for such a purpose module detection methods could still be useful, because pooling across similarly expressed genes might be adventitious given the high levels of noise - and in some cases zero-inflation [78]. Whether this is truly the case should still be assessed.

In this evaluation we found that decomposition methods in particular are suitable for capturing the modular structure of a gene expression dataset. But what then might the future hold for these kind of methods? I think one interesting possibility is the use of deep generative models [79, 80]. These methods learn a representation of a dataset by creating a generative model from a deep learning network. In the case of expression data, the expression is modelled as coming from a limited number of cell cluster nodes and gene module nodes. As noted by the scVI/scANVI preprint, probably the most state-of-the-art deep generative model of single-cell expression data [81]:

> Future principled efforts may focus on putting a suitable prior such as sparsity on neural networks weights. That way, individual neurons of the last hidden layer of the generative model would correspond to individual gene modules, directly readable from the weight sparsity motifs.

Indeed, one recent paper [82] proposes such a method.

Finally, as a meta-research side note, this paper really exemplifies the importance of preprints. Although I generated the main results already within one year of my PhD in 2015, it was only after several rounds of appeals, revisions, journal changes and final journal editing, that the study was finally public March 2018. This was after I was already finishing up the other large benchmarking study (which was put on bioRxiv at the start of March 2018, Chapter 4). During these rounds of revisions, several other studies were published that could have used some of the results of this paper, such as new biclustering methods [83, 84], other evaluation studies [85], applications of methods [86], and reviews [87].

References

[1] M. B. Eisen et al. “Cluster Analysis and Display of Genome-Wide Expression Patterns”. In: *Proceedings of the National Academy of Sciences of the United States of America* 95.25 (Dec. 8, 1998), pp. 14863–14868. ISSN: 0027-8424. DOI: 10.1073/pnas.95.25.14863.

[2] Patrik D’haeseleer. “How Does Gene Expression Clustering Work?” In: *Nature Biotechnology* 23.12 (Dec. 2005), pp. 1499–1501. ISSN: 1087-0156. DOI: 10.1038/nbt1205-1499.

[3] Damien Chaussabel and Nicole Baldwin. “Democratizing Systems Immunology with Modular Transcriptional Repertoire Analyses”. In: *Nature Reviews. Immunology* 14.4 (Apr. 2014), pp. 271–280. ISSN: 1474-1741. DOI: 10.1038/nri3642.

[4] Irina Voineagu et al. “Transcriptomic Analysis of Autistic Brain Reveals Convergent Molecular Pathology”. In: *Nature* 474.7351 (May 25, 2011), pp. 380–384. ISSN: 1476-4687. DOI: 10.1038/nature10110.

[5] Luke Jostins et al. “Host-Microbe Interactions Have Shaped the Genetic Architecture of Inflammatory Bowel Disease”. In: *Nature* 491.7422 (Nov. 1, 2012), pp. 119–124. ISSN: 1476-4687. DOI: 10.1038/nature11582.

[6] Nir Yosef et al. “Dynamic Regulatory Network Controlling TH17 Cell Differentiation”. In: *Nature* 496.7446 (Apr. 25, 2013), pp. 461–468. ISSN: 1476-4687. DOI: 10.1038/nature11981.

[7] Vladimir Jojic et al. “Identification of Transcriptional Regulators in the Mouse Immune System”. In: *Nature Immunology* 14.6 (June 2013), pp. 633–643. ISSN: 1529-2916. DOI: 10.1038/ni.2587.

[8] Franziska Paul et al. “Transcriptional Heterogeneity and Lineage Commitment in Myeloid Progenitors”. In: *Cell* 164.1-2 (Jan. 14, 2016), p. 325. ISSN: 1097-4172. DOI: 10.1016/j.cell.2015.12.046.

[9] Qian Zhu et al. “Targeted Exploration and Analysis of Large Cross-Platform Human Transcriptomic Compendia”. In: *Nature Methods* 12.3 (Mar. 2015), 211–214, 3 p following 214. ISSN: 1548-7105. DOI: 10.1038/nmeth.3249.

[10] Laia Alsina et al. “A Narrow Repertoire of Transcriptional Modules Responsive to Pyogenic Bacteria Is Impaired in Patients Carrying Loss-of-Function Mutations in MYD88 or IRAK4”. In: *Nature Immunology* 15.12 (Dec. 2014), pp. 1134–1142. ISSN: 1529-2916. DOI: 10.1038/ni.3028.

[11] Damien Chaussabel et al. “A Modular Analysis Framework for Blood Genomics Studies: Application to Systemic Lupus Erythematosus”. In: *Immunity* 29.1 (July 18, 2008), pp. 150–164. ISSN: 1097-4180. DOI: 10.1016/j.immuni.2008.05.012.

[12] Shane Neph et al. “Circuitry and Dynamics of Human Transcription Factor Regulatory Networks”. In: *Cell* 150.6 (Sept. 14, 2012), pp. 1274–1286. ISSN: 1097-4172. DOI: 10.1016/j.cell.2012.04.040.

[13] Mark B. Gerstein et al. “Architecture of the Human Regulatory Network Derived from ENCODE Data”. In: *Nature* 489.7414 (Sept. 6, 2012), pp. 91–100. ISSN: 1476-4687. DOI: 10.1038/nature11245.

[14] Andrea Oeckinghaus, Matthew S. Hayden, and Sankar Ghosh. “Crosstalk in NF-κB Signaling Pathways”. In: *Nature Immunology* 12.8 (July 19, 2011), pp. 695–708. ISSN: 1529-2916. DOI: 10.1038/ni.2065.

[15] Daniel Marbach et al. “Wisdom of Crowds for Robust Gene Network Inference”. In: *Nature Methods* 9.8 (July 15, 2012), pp. 796–804. ISSN: 1548-7105. DOI: 10.1038/nmeth.2016.

[16] Maxime Rotival et al. “Integrating Genome-Wide Genetic Variations and Monocyte Expression Data Reveals Trans-Regulated Gene Modules in Humans”. In: *PLoS genetics* 7.12 (Dec. 2011), e1002367. ISSN: 1553-7404. DOI: 10.1371/journal.pgen.1002367.

[17] Kemal Eren et al. “A Comparative Analysis of Biclustering Algorithms for Gene Expression Data”. In: *Briefings in Bioinformatics* 14.3 (May 2013), pp. 279–292. ISSN: 1477-4054. DOI: 10.1093/bib/bbs032.

[18] Sushmita Roy et al. "Integrated Module and Gene-Specific Regulatory Inference Implicates Upstream Signaling Networks". In: *PLoS computational biology* 9.10 (2013), e1003252. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1003252.

[19] Amela Prelić et al. "A Systematic Comparison and Evaluation of Biclustering Methods for Gene Expression Data". In: *Bioinformatics (Oxford, England)* 22.9 (May 1, 2006), pp. 1122–1129. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btl060.

[20] Ali Oghabian et al. "Biclustering Methods: Biological Relevance and Application in Gene Expression Analysis". In: *PloS One* 9.3 (2014), e90801. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0090801.

[21] Anbupalam Thalamuthu et al. "Evaluation and Comparison of Gene Clustering Methods in Microarray Analysis". In: *Bioinformatics (Oxford, England)* 22.19 (Oct. 1, 2006), pp. 2405–2412. ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btl406.

[22] Susmita Datta and Somnath Datta. "Methods for Evaluating Clustering Algorithms for Gene Expression Data Using a Reference Set of Functional Classes". In: *BMC bioinformatics* 7 (Aug. 31, 2006), p. 397. ISSN: 1471-2105. DOI: 10.1186/1471-2105-7-397.

[23] Andrew E. Teschendorff et al. "Elucidating the Altered Transcriptional Programs in Breast Cancer Using Independent Component Analysis". In: *PLoS computational biology* 3.8 (Aug. 2007), e161. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.0030161.

[24] Limin Fu and Enzo Medico. "FLAME, a Novel Fuzzy Clustering Method for the Analysis of DNA Microarray Data". In: *BMC bioinformatics* 8 (Jan. 4, 2007), p. 3. ISSN: 1471-2105. DOI: 10.1186/1471-2105-8-3.

[25] Raquel Norel, John Jeremy Rice, and Gustavo Stolovitzky. "The Self-Assessment Trap: Can We All Be Better than Average?" In: *Molecular Systems Biology* 7 (Oct. 11, 2011), p. 537. ISSN: 1744-4292. DOI: 10.1038/msb.2011.70.

[26] Christian Wiwie, Jan Baumbach, and Richard Röttger. "Comparing the Performance of Biomedical Clustering Methods". In: *Nature Methods* 12.11 (Nov. 2015), pp. 1033–1038. ISSN: 1548-7105. DOI: 10.1038/nmeth.3583.

[27] Stefano Monti et al. "Consensus Clustering: A Resampling-Based Method for Class Discovery and Visualization of Gene Expression Microarray Data". In: *Machine Learning* 52.1 (July 1, 2003), pp. 91–118. ISSN: 1573-0565. DOI: 10.1023/A:1023949509487.

[28] Olatz Arbelaitz et al. "An Extensive Comparative Study of Cluster Validity Indices". In: *Pattern Recognition* 46.1 (Jan. 1, 2013), pp. 243–256. ISSN: 0031-3203. DOI: 10.1016/j.patcog.2012.07.021.

[29] Ido Priness, Oded Maimon, and Irad Ben-Gal. "Evaluation of Gene-Expression Clustering via Mutual Information Distance Measure". In: *BMC bioinformatics* 8 (Mar. 30, 2007), p. 111. ISSN: 1471-2105. DOI: 10.1186/1471-2105-8-111.

[30] Andy M. Yip and Steve Horvath. "Gene Network Interconnectedness and the Generalized Topological Overlap Measure". In: *BMC bioinformatics* 8 (Jan. 24, 2007), p. 22. ISSN: 1471-2105. DOI: 10.1186/1471-2105-8-22.

[31] Lin Song, Peter Langfelder, and Steve Horvath. "Comparison of Co-Expression Measures: Mutual Information, Correlation, and Model Based Indices". In: *BMC bioinformatics* 13 (Dec. 9, 2012), p. 328. ISSN: 1471-2105. DOI: 10.1186/1471-2105-13-328.

[32] Konrad J. Karczewski et al. "Coherent Functional Modules Improve Transcription Factor Target Identification, Cooperativity Prediction, and Disease Association". In: *PLoS genetics* 10.2 (Feb. 2014), e1004122. ISSN: 1553-7404. DOI: 10.1371/journal.pgen.1004122.

[33] Riet De Smet and Kathleen Marchal. "Advantages and Limitations of Current Network Inference Methods". In: *Nature Reviews. Microbiology* 8.10 (Oct. 2010), pp. 717–729. ISSN: 1740-1534. DOI: 10.1038/nrmicro2419.

[34] Peter Langfelder and Steve Horvath. "WGCNA: An R Package for Weighted Correlation Network Analysis". In: *BMC bioinformatics* 9 (Dec. 29, 2008), p. 559. ISSN: 1471-2105. DOI: 10.1186/1471-2105-9-559.

[35] Marc Streit et al. “Furby: Fuzzy Force-Directed Bicluster Visualization”. In: *BMC bioinformatics* 15 Suppl 6 (2014), S4. ISSN: 1471-2105. DOI: 10.1186/1471-2105-15-S6-S4.

[36] Rodrigo Santamaría, Roberto Therón, and Luis Quintales. “BicOverlapper 2.0: Visual Analysis for Gene Expression”. In: *Bioinformatics (Oxford, England)* 30.12 (June 15, 2014), pp. 1785–1786. ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btu120.

[37] Bie Verbist et al. “Using Transcriptomics to Guide Lead Optimization in Drug Discovery Projects: Lessons Learned from the QSTAR Project”. In: *Drug Discovery Today* 20.5 (May 2015), pp. 505–513. ISSN: 1878-5832. DOI: 10.1016/j.drudis.2014.12.014.

[38] Heladia Salgado et al. “RegulonDB v8.0: Omics Data Sets, Evolutionary Conservation, Regulatory Phrases, Cross-Validated Gold Standards and More”. In: *Nucleic Acids Research* 41.D1 (Jan. 1, 2013), pp. D203–D213. ISSN: 0305-1048. DOI: 10.1093/nar/gks1201.

[39] Sisi Ma et al. “De-Novo Learning of Genome-Scale Regulatory Networks in S. Cerevisiae”. In: *PloS One* 9.9 (2014), e106479. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0106479.

[40] Kenzie D. MacIsaac et al. “An Improved Map of Conserved Regulatory Sites for Saccharomyces Cerevisiae”. In: *BMC bioinformatics* 7 (Mar. 7, 2006), p. 113. ISSN: 1471-2105. DOI: 10.1186/1471-2105-7-113.

[41] Daniel Marbach et al. “Tissue-Specific Regulatory Circuits Reveal Variable Modular Perturbations across Complex Diseases”. In: *Nature Methods* 13.4 (Apr. 2016), pp. 366–370. ISSN: 1548-7105. DOI: 10.1038/nmeth.3799.

[42] S.M. van Dongen et al. “Graph Clustering by Flow Simulation”. In: (Feb. 2001).

[43] Tobias Wittkop et al. “Partitioning Biological Data with Transitivity Clustering”. In: *Nature Methods* 7.6 (June 2010), pp. 419–420. ISSN: 1548-7105. DOI: 10.1038/nmeth0610-419.

[44] Brendan J. Frey and Delbert Dueck. “Clustering by Passing Messages between Data Points”. In: *Science (New York, N.Y.)* 315.5814 (Feb. 16, 2007), pp. 972–976. ISSN: 1095-9203. DOI: 10.1126/science.1136800.

[45] “Gene Ontology Consortium: Going Forward”. In: *Nucleic Acids Research* 43.D1 (Jan. 28, 2015), pp. D1049–D1056. ISSN: 0305-1048. DOI: 10.1093/nar/gku1179.

[46] Minoru Kanehisa et al. “KEGG as a Reference Resource for Gene and Protein Annotation”. In: *Nucleic Acids Research* 44.D1 (Jan. 4, 2016), pp. D457–462. ISSN: 1362-4962. DOI: 10.1093/nar/gkv1070.

[47] Pieter Meysman et al. “COLOMBOS v2.0: An Ever Expanding Collection of Bacterial Expression Compendia”. In: *Nucleic Acids Research* 42 (Database issue Jan. 2014), pp. D649–653. ISSN: 1362-4962. DOI: 10.1093/nar/gkt1086.

[48] Katherine A. Hoadley et al. “Multiplatform Analysis of 12 Cancer Types Reveals Molecular Classification within and across Tissues of Origin”. In: *Cell* 158.4 (Aug. 14, 2014), pp. 929–944. ISSN: 1097-4172. DOI: 10.1016/j.cell.2014.06.049.

[49] “The Genotype-Tissue Expression (GTEx) Pilot Analysis: Multitissue Gene Regulation in Humans”. In: *Science (New York, N.Y.)* 348.6235 (May 8, 2015), pp. 648–660. ISSN: 0036-8075. DOI: 10.1126/science.1262110.

[50] Thomas Schaffter, Daniel Marbach, and Dario Floreano. “GeneNetWeaver: In Silico Benchmark Generation and Performance Profiling of Network Inference Methods”. In: *Bioinformatics (Oxford, England)* 27.16 (Aug. 15, 2011), pp. 2263–2270. ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btr373.

[51] Richard Röttger et al. “How Little Do We Actually Know? On the Size of Gene Regulatory Networks”. In: *IEEE/ACM transactions on computational biology and bioinformatics* 9.5 (2012), pp. 1293–1300. ISSN: 1557-9964. DOI: 10.1109/TCBB.2012.71.

[52] François Spitz and Eileen E. M. Furlong. “Transcription Factors: From Enhancer Binding to Developmental Control”. In: *Nature Reviews. Genetics* 13.9 (Sept. 2012), pp. 613–626. ISSN: 1471-0064. DOI: 10.1038/nrg3207.

[53] Sepp Hochreiter et al. “FABIA: Factor Analysis for Bicluster Acquisition”. In: *Bioinformatics (Oxford, England)* 26.12 (June 15, 2010), pp. 1520–1527. ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btq227.

[54] Vanessa Vermeirssen et al. “Arabidopsis Ensemble Reverse-Engineered Gene Regulatory Network Discloses Interconnected Transcription Factors in Oxidative Stress”. In: *The Plant Cell* 26.12 (Dec. 2014), pp. 4656–4679. ISSN: 1532-298X. DOI: 10.1105/tpc.114.131417.

[55] Archana Ramesh et al. “Clustering Context-Specific Gene Regulatory Networks.” In: *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing* (Jan. 2010), pp. 444–55. ISSN: 2335-6936.

[56] Vladimir Estivill-Castro. “Why so Many Clustering Algorithms”. In: *ACM SIGKDD Explorations Newsletter* 4.1 (June 2002), pp. 65–75. ISSN: 19310145. DOI: 10.1145/568574.568575.

[57] L Lazzeroni and A Owen. “Plaid Models for Gene Expression Data”. In: *Statistica sinica* (2002).

[58] Amir Ben-Dor et al. “Discovering Local Structure in Gene Expression Data: The Order-Preserving Submatrix Problem”. In: *Journal of Computational Biology: A Journal of Computational Molecular Cell Biology* 10.3-4 (2003), pp. 373–384. ISSN: 1066-5277. DOI: 10.1089/10665270360688075.

[59] Enrique Amigó et al. “A Comparison of Extrinsic Clustering Evaluation Metrics Based on Formal Constraints”. In: *Information Retrieval* 12.4 (July 2008), pp. 461–486. ISSN: 1386-4564. DOI: 10.1007/s10791-008-9066-8.

[60] Henry Rosales-Méndez and Yunior Ramírez-Cruz. “CICE-BCubed: A New Evaluation Measure for Overlapping Clustering Algorithms”. In: *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Application.* Vol. 8258. Lecture Notes in Computer Science. Springer, Heidelberg, Germany, 2013, pp. 157–164. ISBN: 978-3-642-41821-1. DOI: 10.1007/978-3-642-41822-8\{\\_\}20.

[61] Natasa Przulj and Desmond J. Higham. “Modelling Protein-Protein Interaction Networks via a Stickiness Index”. In: *Journal of the Royal Society, Interface* 3.10 (Oct. 22, 2006), pp. 711–716. ISSN: 1742-5689. DOI: 10.1098/rsif.2006.0147.

[62] B Bollobás et al. “Directed Scale-Free Graphs”. In: *Proceedings of the* (2003).

[63] Stanton A. Glantz. *Primer of Biostatistics.* McGraw-Hill Medical Pub, 2005. 520 pp. ISBN: 0-07-143509-3.

[64] David N. Reshef et al. “Detecting Novel Associations in Large Data Sets”. In: *Science (New York, N.Y.)* 334.6062 (Dec. 16, 2011), pp. 1518–1524. ISSN: 1095-9203. DOI: 10.1126/science.1205438.

[65] Helen Shen. “Interactive Notebooks: Sharing the Code”. In: *Nature* 515.7525 (Nov. 6, 2014), pp. 151–152. ISSN: 1476-4687. DOI: 10.1038/515151a.

[66] F. Perez and B. E. Granger. “IPython: A System for Interactive Scientific Computing”. In: *Computing in Science Engineering* 9.3 (May 2007), pp. 21–29. ISSN: 1521-9615. DOI: 10.1109/MCSE.2007.53.

[67] Nima Aghaeepour et al. “Critical Assessment of Automated Flow Cytometry Data Analysis Techniques.” In: *Nature methods* 10.3 (Mar. 2013), pp. 228–38. ISSN: 1548-7105. DOI: 10.1038/nmeth.2365.

[68] E Rendón and I Abundez. “Internal versus External Cluster Validation Indexes”. In: *International Journal of ...* (2011).

[69] A Lancichinetti, S Fortunato, and J Kertész. “Detecting the Overlapping and Hierarchical Community Structure in Complex Networks”. In: *New Journal of Physics* (2009).

[70] MK Mark K. Goldberg, Mykola Hayvanovych, and Malik Magdon-Ismail. “Measuring Similarity between Sets of Overlapping Clusters”. In: *2010 IEEE Second International Conference on Social Computing.* IEEE, Aug. 2010, pp. 303–308. ISBN: 978-1-4244-8439-3. DOI: 10.1109/SocialCom.2010.50.

[71] Heather Turner, Trevor Bailey, and Wojtek Krzanowski. “Improved Biclustering of Microarray Data Demonstrated through Systematic Performance Tests”. In: *Comput. Stat. Data Anal.* 48.2 (2005), pp. 235–254.

[72] Breck Baldwin Amit Bagga. "Entity-Based Cross-Document Coreferencing Using the Vector Space Model". In: *Proceedings of the 17th International Conference on Computational Linguistics*. Association for Computational Linguistics, 1998, pp. 79–85. DOI: 10.3115/980451.980859.

[73] Peter Langfelder et al. "Is My Network Module Preserved and Reproducible?" In: *PLoS computational biology* 7.1 (Jan. 2011), e1001057. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1001057.

[74] Guojun Li et al. "{QUBIC}: A Qualitative Biclustering Algorithm for Analyses of Gene Expression Data". In: *Nucleic Acids Res.* 37.15 (Aug. 2009), e101.

[75] Juan Xie et al. "QUBIC2: A Novel Biclustering Algorithm for Large-Scale Bulk RNA-Sequencing and Single-Cell RNA-Sequencing Data Analysis". In: *bioRxiv* (Sept. 7, 2018), p. 409961. DOI: 10.1101/409961.

[76] Aviv Regev et al. "Science Forum: The Human Cell Atlas". In: *eLife* 6 (Dec. 5, 2017), e27041. ISSN: 2050-084X. DOI: 10.7554/eLife.27041.

[77] Xiaoping Han et al. "Mapping the Mouse Cell Atlas by Microwell-Seq". In: *Cell* 172.5 (Feb. 22, 2018), 1091–1107.e17. ISSN: 1097-4172. DOI: 10.1016/j.cell.2018.02.001.

[78] Justin D. Silverman et al. "Naught All Zeros in Sequence Count Data Are the Same". In: *bioRxiv* (Nov. 26, 2018), p. 477794. DOI: 10.1101/477794.

[79] Romain Lopez et al. "Deep Generative Modeling for Single-Cell Transcriptomics". In: *Nature Methods* 15.12 (Dec. 2018), p. 1053. ISSN: 1548-7105. DOI: 10.1038/s41592-018-0229-2.

[80] Jiarui Ding, Anne Condon, and Sohrab P. Shah. "Interpretable Dimensionality Reduction of Single Cell Transcriptome Data with Deep Generative Models". In: *Nature Communications* 9.1 (May 21, 2018), p. 2002. ISSN: 2041-1723. DOI: 10.1038/s41467-018-04368-5.

[81] Chenling Xu et al. "Harmonization and Annotation of Single-Cell Transcriptomics Data with Deep Generative Models". In: *bioRxiv* (Jan. 29, 2019), p. 532895. DOI: 10.1101/532895.

[82] Tian Tian et al. "Clustering Single-Cell RNA-Seq Data with a Model-Based Deep Learning Approach". In: *Nature Machine Intelligence* 1.4 (Apr. 2019), p. 191. ISSN: 2522-5839. DOI: 10.1038/s42256-019-0037-0.

[83] Anindya Bhattacharya and Yan Cui. "A GPU-Accelerated Algorithm for Biclustering Analysis and Detection of Condition-Dependent Coexpression Network Modules". In: *Scientific Reports* 7.1 (June 23, 2017), p. 4162. ISSN: 2045-2322. DOI: 10.1038/s41598-017-04070-4.

[84] Peng Qiu. "Embracing the Dropouts in Single-Cell RNA-Seq Data". In: *bioRxiv* (Nov. 17, 2018), p. 468025. DOI: 10.1101/468025.

[85] Victor A. Padilha and Ricardo J. G. B. Campello. "A Systematic Comparative Evaluation of Biclustering Techniques". In: *BMC Bioinformatics* 18.1 (Jan. 23, 2017), p. 55. ISSN: 1471-2105. DOI: 10.1186/s12859-017-1487-1.

[86] Amit Zeisel et al. "Brain Structure. Cell Types in the Mouse Cortex and Hippocampus Revealed by Single-Cell RNA-Seq". In: *Science (New York, N.Y.)* 347.6226 (Mar. 6, 2015), pp. 1138–1142. ISSN: 1095-9203. DOI: 10.1126/science.aaa1934.

[87] Juan Xie et al. "It Is Time to Apply Biclustering: A Comprehensive Review of Biclustering Applications in Biological and Biomedical Data". In: *Briefings in Bioinformatics* (). DOI: 10.1093/bib/bby014.

4 | Comparing trajectory inference methods

The idea behind trajectory inference methods is simple: if the expression of a cell changes gradually during a dynamic process (such as differentiation), we can find common paths that cells take by connecting cells with similar expression profiles. When I first read the landmark papers describing these methods, I was primarily impressed by the possibilities they may offer in immunology. Immune cells constantly change and adapt, both while differentiating, steady-state and during an immune response. If you could reconstruct these dynamics, it would give us tremendous insights into their development and function, and what happens during diseases.

However, we quickly found out that this was easier said then done. Methods often required prior information such as a starting cell, which would introduce a bias. Moreover, a lot of methods performed poorly on our datasets, or were too sensitive to outliers for them to be useful. Together with a colleague PhD student, Robrecht Cannoodt, we thus began developing a benchmarking pipeline to compare these methods, and try to improve them along the way.

I have embedded the most relevant supplementary material directly within this thesis, including Supplementary Note 1. Other supplementary material is available at https://www.nature.com/articles/s41587-019-0071-9#Sec34

At the end of this chapter, I discuss some issues that have popped up after the publication of this paper (section Update).

A comparison of single-cell trajectory inference methods

Published in Nature Biotechnology, March 15 2018, doi.org/gfxsgd
Wouter Saelens*, Robrecht Cannoodt*, Helena Todorov, Yvan Saeys
*: Equal contribution

Contributions
The design, execution and writing of this work was completely shared with Robrecht Cannoodt. Given the intense collaboration, it is difficult to say which parts of the work were done by me, because of frequent redesigning of each other's figures, refactoring of each other's code and implementation of each other's ideas.

Designed the study: **Wouter Saelens**, Robrecht Cannoodt, Helena Todorov and Yvan Saeys
Performed the experiments and analysed the data: **Wouter Saelens** and Robrecht Cannoodt
Implemented software packages: **Wouter Saelens**, Robrecht Cannoodt and Helena Todorov
Prepared the manuscript: **Wouter Saelens**, Robrecht Cannoodt, Helena Todorov and Yvan Saeys
Supervised the work: Yvan Saeys

Abstract
Trajectory inference approaches analyse 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 analyse increasingly large and complex single-cell datasets.

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 [1, 2]. 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 [3, 4, 5]. The resulting trajectories are most often linear, bifurcating or tree-shaped, but more recent methods also identify more complex trajectory topologies, such as cyclic [6] or disconnected graphs [7]. TI methods offer an unbiased and transcriptome-wide understanding of a dynamic process [1], thereby allowing the objective identification of new (primed) subsets of cells [8], delineation of a differentiation tree [9, 10] and inference of regulatory interactions responsible for one or more bifurcations [11]. Current applications of TI focus on specific subsets of cells, but ongoing efforts to construct transcriptomic catalogs of whole organisms [12, 13, 14] underline the urgency for accurate, scalable [11, 15] 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 [16], the 'awesome-single-cell' list [17] and scRNA-tools.org [18], 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 [19, 8, 20, 21] or bifurcating trajectories [22, 23]) or through parameters provided by the user [24, 25]. These methods therefore mainly focus on correctly ordering the cells along the fixed topology. More recent methods also infer the topology [26, 27, 7], 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 [22, 28, 29, 25, 30, 31, 32, 33, 7], 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 4.1a). 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.

Results

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 4.1b). In this 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 4.2) and we wrote a common output converter for each of these groups (Figure 4.3a). 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 4.3a).

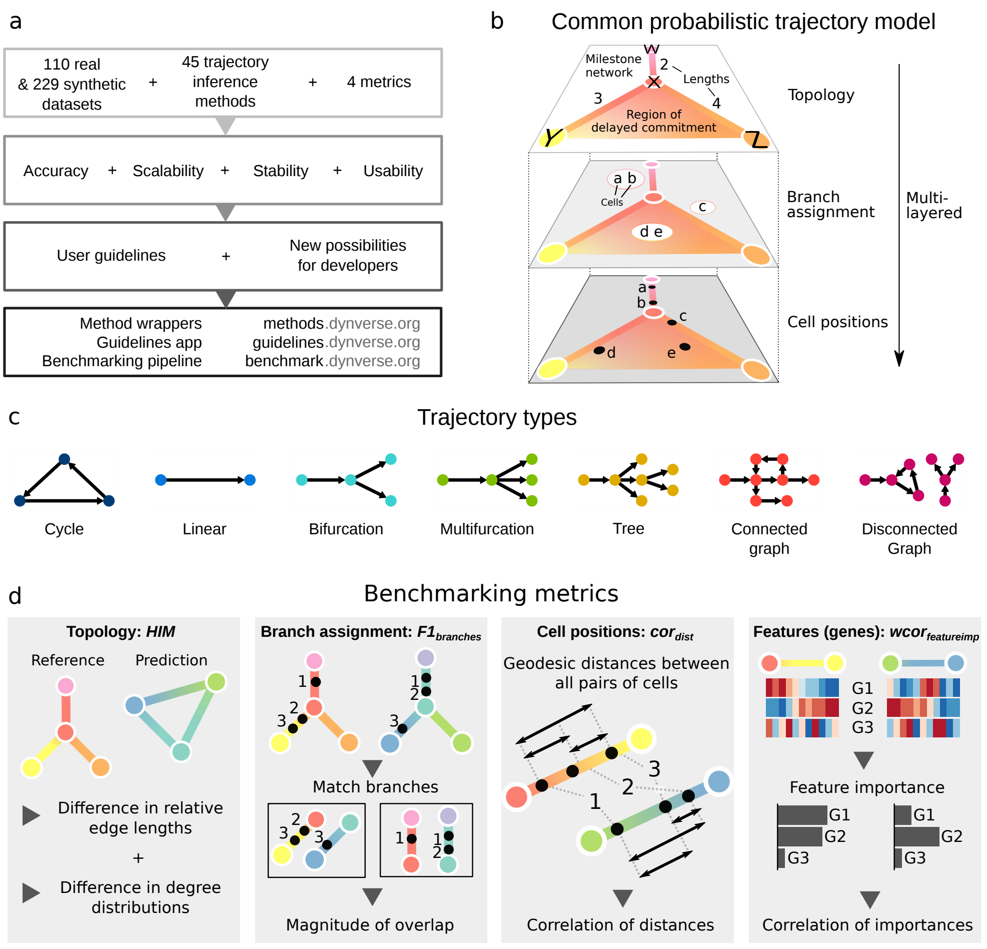


Figure 4.1: Overview of several key aspects of the evaluation. **a**, A schematic overview of our evaluation pipeline. **b**, To make the trajectories comparable to each other, a common trajectory model was used to represent reference trajectories from the real and synthetic datasets, as well as any 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 $F1_{branches}$ assesses the similarity of the assignment of cells onto branches. The cor_{dist} quantifies the similarity in cellular positions between two trajectories, by calculating the correlation between pairwise geodesic distances. Finally, $wcor_{features}$ quantifies the agreement between trajectory differentially expressed features from the known trajectory and the predicted trajectory.

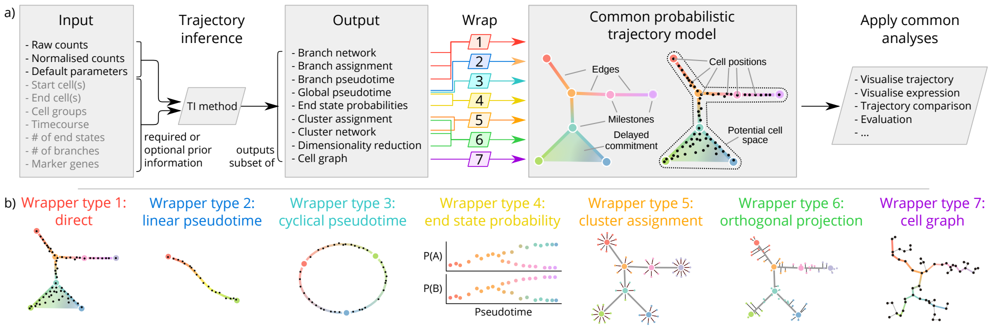
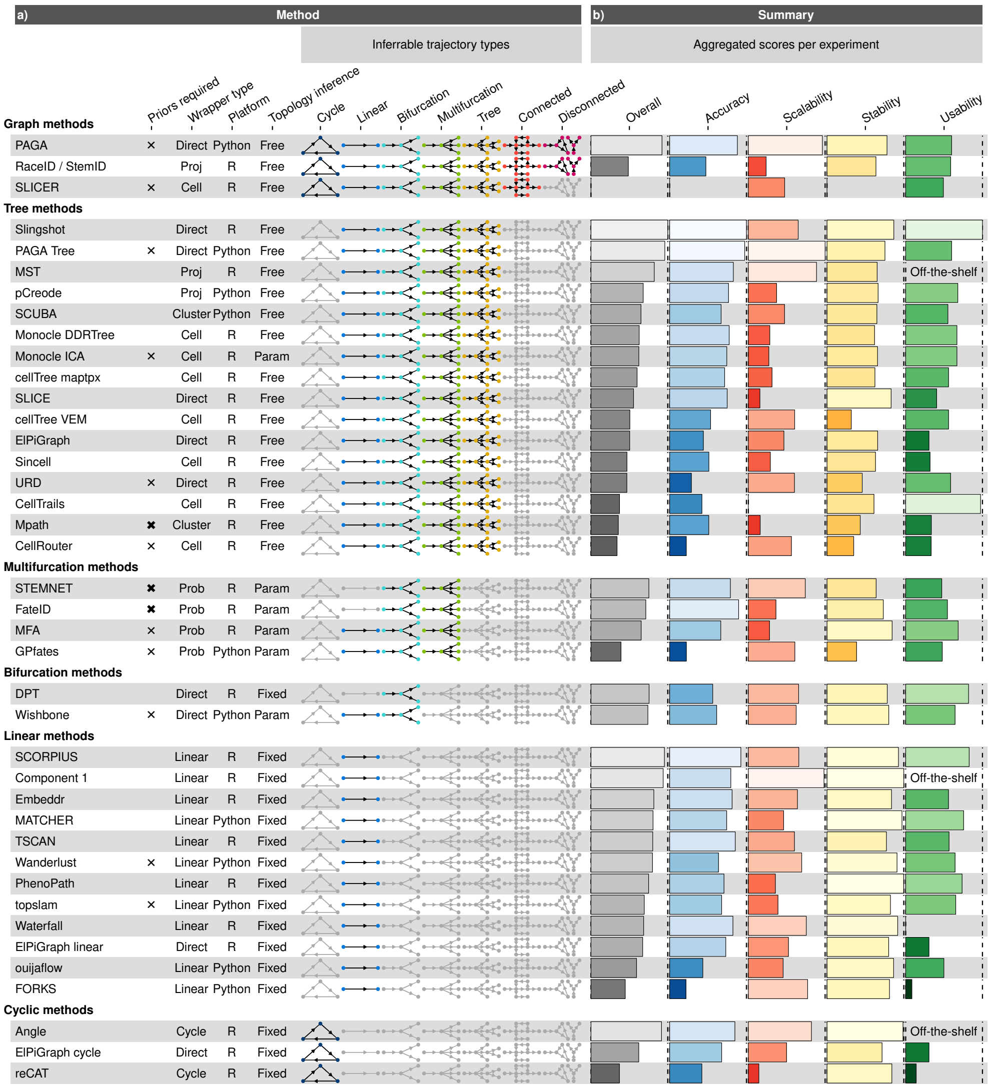


Figure 4.2: A common interface for TI methods. **a** The input and output of each TI method is standardised. As input, each TI method receives either raw or normalised counts, several parameters, and a selection of prior information. After its execution, a method uses one of the seven wrapper functions 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.



	Priors required	Wrapper type	Platform	Topology inference
Graph methods				
PAGA	×	Direct	Python	Free
RaceID / StemID		Proj	R	Free
SLICER	×	Cell	R	Free
Tree methods				
Slingshot		Direct	R	Free
PAGA Tree	×	Direct	Python	Free
MST		Proj	R	Free
pCreode		Proj	Python	Free
SCUBA		Cluster	Python	Free
Monocle DDRTree		Cell	R	Free
Monocle ICA	×	Cell	R	Param
cellTree maptpx		Cell	R	Free
SLICE		Direct	R	Free
cellTree VEM		Cell	R	Free
ElPiGraph		Direct	R	Free
Sincell		Cell	R	Free
URD	×	Direct	R	Free
CellTrails		Cell	R	Free
Mpath	✖	Cluster	R	Free
CellRouter	×	Cell	R	Free
Multifurcation methods				
STEMNET	✖	Prob	R	Param
FateID	✖	Prob	R	Param
MFA	×	Prob	R	Param
GPfates	×	Prob	Python	Param
Bifurcation methods				
DPT		Direct	R	Fixed
Wishbone	×	Direct	Python	Param
Linear methods				
SCORPIUS		Linear	R	Fixed
Component 1		Linear	R	Fixed
Embeddr		Linear	R	Fixed
MATCHER		Linear	Python	Fixed
TSCAN		Linear	R	Fixed
Wanderlust	×	Linear	Python	Fixed
PhenoPath		Linear	R	Fixed
topslam	×	Linear	Python	Fixed
Waterfall		Linear	R	Fixed
ElPiGraph linear		Direct	R	Fixed
ouijaflow		Linear	Python	Fixed
FORKS		Linear	Python	Fixed
Cyclic methods				
Angle		Cycle	R	Fixed
ElPiGraph cycle		Direct	R	Fixed
reCAT		Cycle	R	Fixed

Prior information required
- None
- × Weak: Start or end cells
- ✖ Strong: Cell grouping or time course

Not shown, insufficient data points

CALISTA	ouija
cellTree Gibbs	pseudogp
GrandPrix	SCIMITAR
MERLoT	SCOUP

Figure caption on next page →

Figure 4.3: A characterization of the 45 methods evaluated in this study and their overall evaluation results. **a**, We characterised 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 method usability. Methods that errored on more than 50% of the datasets are not included in this figure and are shown instead in Supplementary Fig. 2.

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 4.3a).

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 4.3b). We will discuss each evaluation criterion in more detail (Figure 4.4 and Supplementary Fig. 2), after which we conclude with guidelines for method users and future perspectives for method developers.

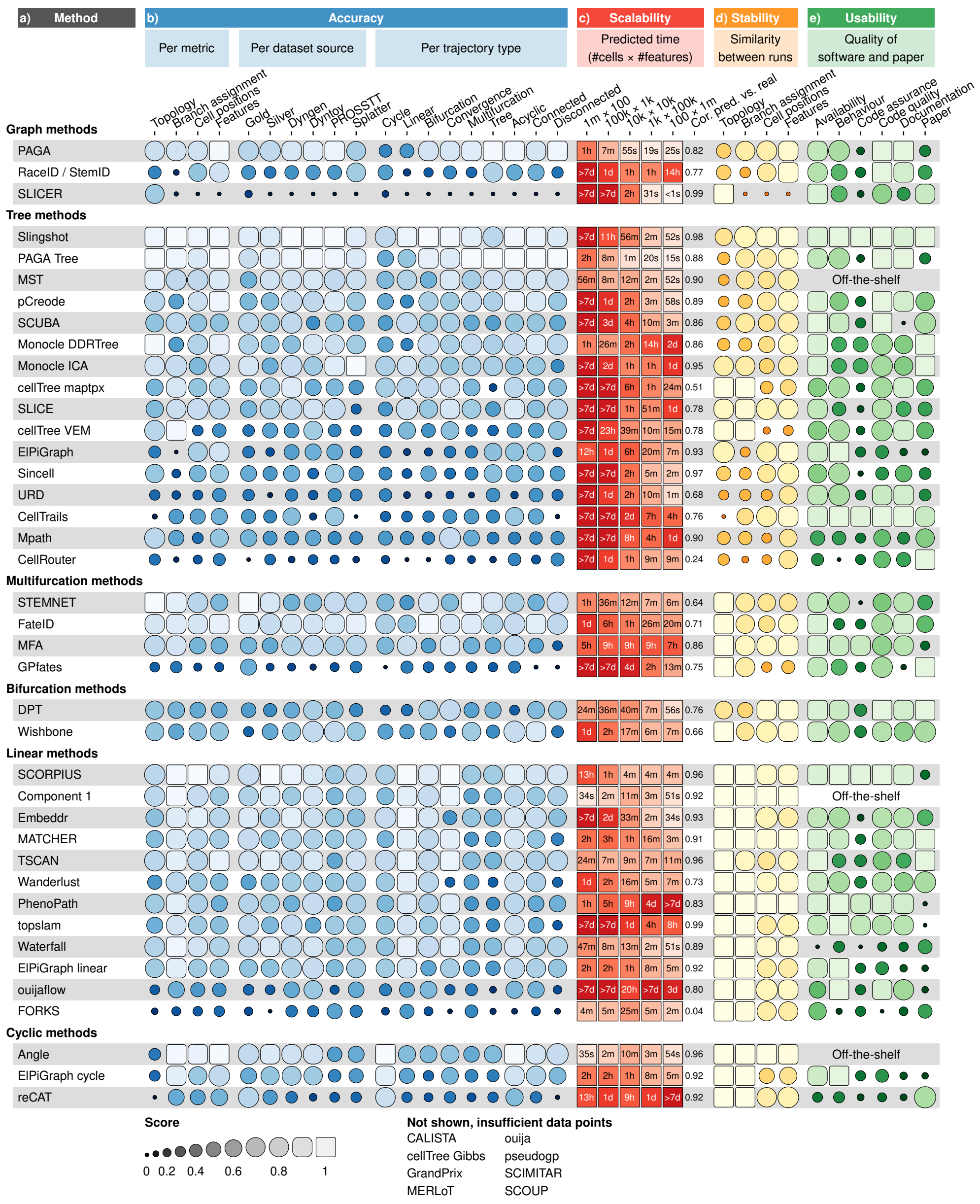


Figure caption on next page →

Figure 4.4: Detailed results of the four main evaluation criteria: accuracy, scalability, stability and usability. a, The names of the methods, ordered as in Figure 4.3. **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.

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 4.1d): the topology (Hamming–Ipsen–Mikhailov, *HIM*), the quality of the assignment of cells to branches ($F1_{branches}$), the cell positions (cor_{dist}) and the accuracy of the differentially expressed features along the trajectory ($wcor_{features}$). The data compendium consisted of both synthetic datasets, which offer the most 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 [34]. 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 [35]. 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 [36].

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 4.5a). 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 4.4b). 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 4.5b), 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 4.4b).

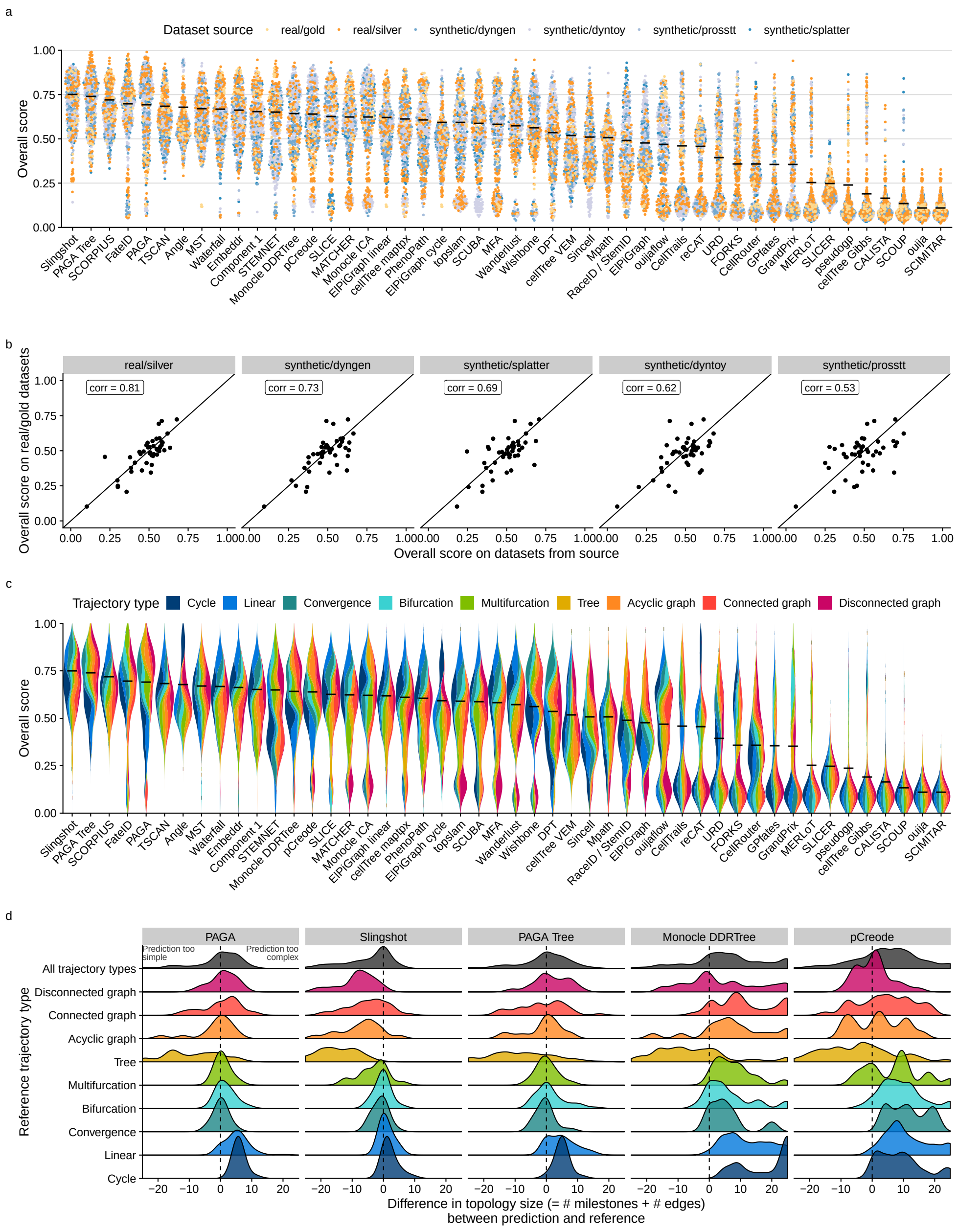


Figure caption on next page →

Figure 4.5: 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.

The performance of a method was strongly dependent on the type of trajectory present in the data (Figure 4.4b). 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 4.5c). 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 4.5d). 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 4.6a). 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 4.6b), 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 utilise this complementarity.

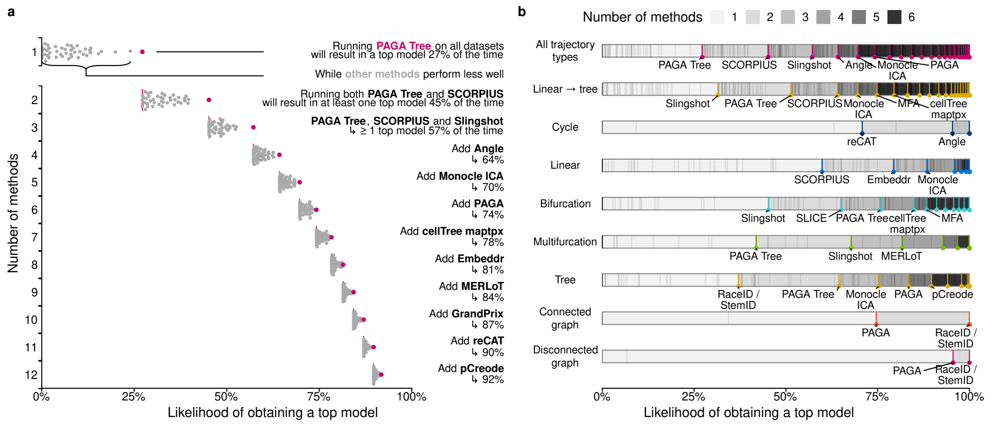


Figure 4.6: Complementarity between different trajectory inference methods. **a**, We assessed the likelihood for different combinations of methods to lead to a 'top model' (defined as a model with an overall score of at least 95% of the best model) when applied to all datasets. **b**, The likelihood 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.

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 more than ten million [37]. Moreover, the recent application of TI methods on multi-omics single-cell data also showcases the increasing demands on the number of features [38]. 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 [39] (Figure 4.7a). 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 4.4c 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 4.4c), which is around the size of a typical droplet-based single-cell dataset [37]. Running times increased further with increasing number of cells, with only a handful of graph/tree methods completing

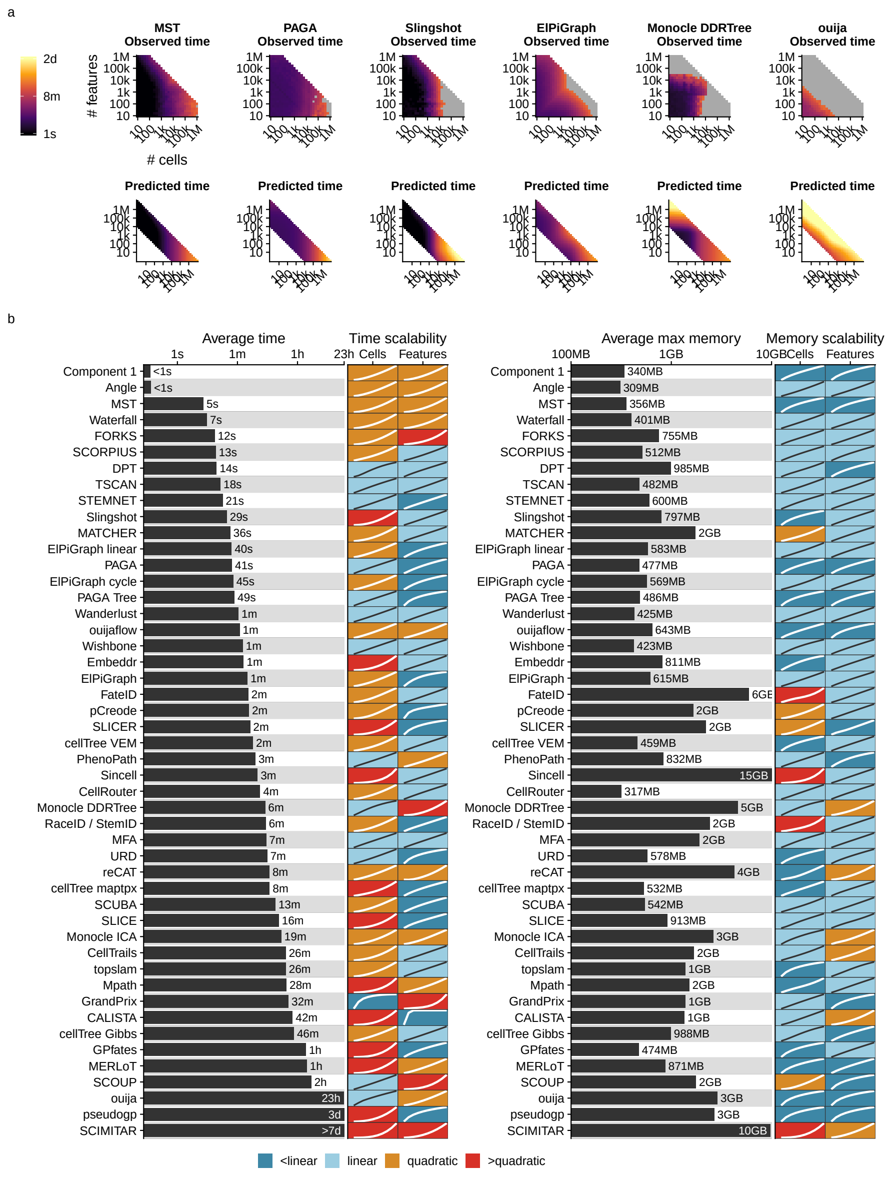


Figure 4.7: Scalability of trajectory inference methods. **a** Three examples of average observed running times across five datasets (left) and the predicted running time (right). **b** Overview of the scalability results of all methods, ordered by their average predicted running time from (a). We predicted 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.

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 4.7b). 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 4.7b).

We also assessed the memory requirements of each method (Supplementary Fig. 2c). Most methods had reasonable memory requirements for modern workstations or computer clusters (≤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).

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.

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 4.4d).

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.

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 [40]. 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 4.1). 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:unwrappable). Installation issues seem to be quite general in bioinformatics [41] 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 4.4d 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 [42, 43]. 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 4.3b).

Table 4.1: 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	[44, 42, 40, 45, 43, 46, 47]
Version control	The code is available on a public version controlled repository, such as Github	[44, 42, 40, 45, 43, 46]
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, …	[44, 45, 47, 46]
Dependencies	(1) Dependencies are clearly stated in the tutorial or in the code (2) Dependencies are automatically installed	[40, 45, 43, 48]
Licence	(1) The code is licensed (2) Licence allows academic use	[44, 40, 45, 43, 46, 47]
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	[46]
Code quality		
Function and object naming	(1) Functions/commands have well chosen names (2) Arguments/parameters have well chosen names	[42, 45]
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	[42, 45, 43]
Code duplication	Duplicated code is minimal	[42, 45]

Self-contained functions	The method is exposed to the user as self-contained functions or commands	[49, 40, 46]
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	[44, 48]
Code assurance		
Unit testing	Method is tested using unit tests	[44, 42, 49, 45, 46]
Unit testing	Tests are run automatically using functionality from the programming language	[44, 42, 49, 45, 46]
Continuous integration	The method uses continuous integration, for example on Travis CI	[50, 45, 43, 46]
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	[42, 45, 43, 46, 47]
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.	[51]
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)	[45, 46, 47, 48, 52]
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	[42, 40, 45, 46, 48]
Inline documentation	Inline documentation is present in the code	[42, 40, 45, 46, 48]
Parameter transparency	All important parameters are exposed to the user	[40]
Behaviour		
Seed setting	The method does not artificially become deterministic, for example by setting some (0.5) or a lot (1) of seeds	[53]
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	[43]
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)	

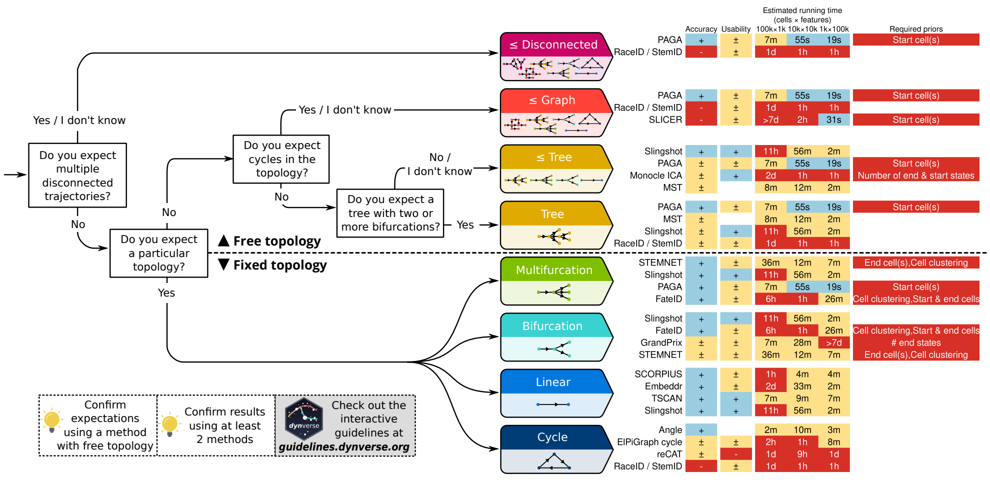
Paper		
Publishing	The method is published	
Peer review	The paper is published in a peer-reviewed journal	[48, 54, 55]
Evaluation on real data	(1) The paper shows the method's usefulness on several (1), one (0.25) or no real datasets. (2) The paper quantifies the accuracy of the method given a gold or silver standard trajectory	[56, 57]
Evaluation of robustness	The paper assessed method robustness (to eg. noise, subsampling, parameter changes, stability) in one (0.5) or several (1) ways	[48, 56, 52, 57]

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.

Based on the results of our benchmark, we propose a set of practical guidelines for method users (Figure 4.8 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 4.8. 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 (https://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 set-


Do you expect multiple disconnected trajectories?
Yes / I don't know
No
Do you expect a particular topology?
No
Yes
Do you expect cycles in the topology?
Yes / I don't know
No
Do you expect a tree with two or more bifurcations?
No / I don't know
Yes
▲ Free topology
▼ Fixed topology
≤ Disconnected
≤ Graph
≤ Tree
Tree
Multifurcation
Bifurcation
Linear
Cycle
Confirm expectations using a method with free topology
Confirm results using at least 2 methods
Check out the interactive guidelines at guidelines.dynverse.org
dynverse

				Estimated running time (cells × features)			
		Accuracy	Usability	100k×1k	10k×10k	1k×100k	Required priors
≤ Disconnected	PAGA	+	±	7m	55s	19s	Start cell(s)
	RaceID / StemID	-	±	1d	1h	1h	
≤ Graph	PAGA	+	±	7m	55s	19s	Start cell(s)
	RaceID / StemID	-	±	1d	1h	1h	
	SLICER	-	±	>7d	2h	31s	Start cell(s)
≤ Tree	Slingshot	+	+	11h	56m	2m	
	PAGA	±	±	7m	55s	19s	Start cell(s)
	Monocle ICA	±	+	2d	1h	1h	Number of end & start states
	MST	±		8m	12m	2m	
Tree	PAGA	+	±	7m	55s	19s	Start cell(s)
	MST	±		8m	12m	2m	
	Slingshot	±	+	11h	56m	2m	
	RaceID / StemID	±	±	1d	1h	1h	
Multifurcation	STEMNET	+	±	36m	12m	7m	End cell(s),Cell clustering
	Slingshot	+	+	11h	56m	2m	
	PAGA	+	±	7m	55s	19s	Start cell(s)
	FateID	+	±	6h	1h	26m	Cell clustering,Start & end cells
Bifurcation	Slingshot	+	+	11h	56m	2m	
	FateID	+	±	6h	1h	26m	Cell clustering,Start & end cells
	GrandPrix	±	±	7m	28m	>7d	# end states
	STEMNET	±	±	36m	12m	7m	End cell(s),Cell clustering
Linear	SCORPIUS	+	±	1h	4m	4m	
	Embeddr	+	±	2d	33m	2m	
	TSCAN	+	+	7m	9m	7m	
	Slingshot	+	+	11h	56m	2m	
Cycle	Angle	+		2m	10m	3m	
	ElPiGraph cycle	±	±	2h	1h	8m	
	reCAT	±	-	1d	9h	1d	
	RaceID / StemID	-	±	1d	1h	1h	

Figure 4.8: 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 ≥0.9, ±: >0.6), usability scores (+:≥0.9, ± ≥0.6), estimated running times and required prior information. k, thousands; m, millions.

ting 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, compare different predicted trajectories and apply downstream analyses, such as finding genes important for the trajectory, network inference [11] or finding modules of genes [58]. 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 4.9). 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 [59], 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.

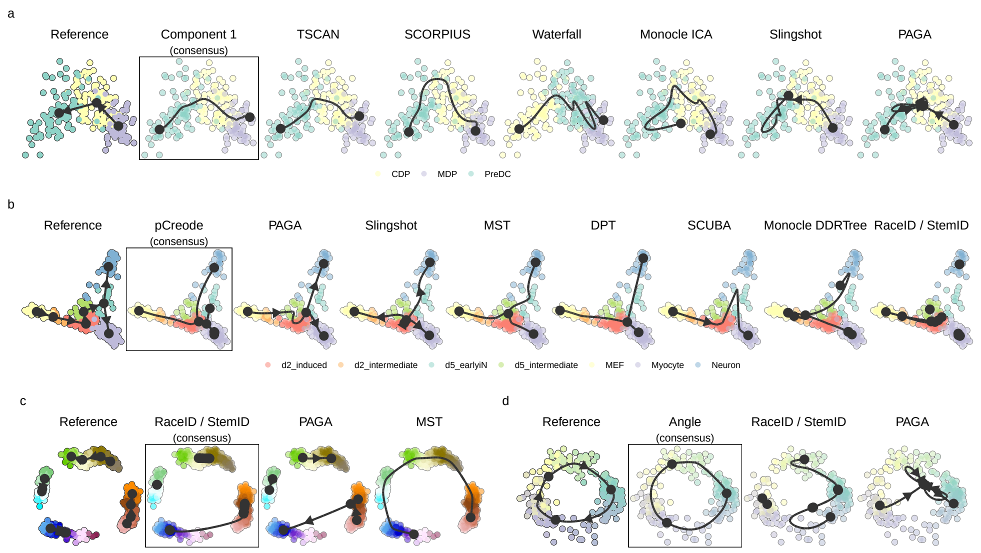


Figure 4.9: Demonstration of how a common framework for TI methods facilitates broad applicability using some example datasets. Trajectories inferred by each method were projected to a common dimensionality reduction using multidimensional scaling. For each dataset, we also calculated a 'consensus' prediction, by calculating the cor_{dist} 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.

Our study indicates that the field of trajectory inference is maturing, primarily for linear and bifurcating trajectories (Figure 4.9a,b). However, we also highlight several ongoing challenges, which should be addressed before TI can be a reliable tool for analysing 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 4.9c,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 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 [60] 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.

Methods

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 [17] and https://github.com/agitter/single-cell-pseudotime [61]. 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 × 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.

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 normalised 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 uniformily discretised 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 4.2a). 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 4.2b). 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×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 ≠ 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 [62, 63].

- **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.

Trajectory types

We classified all possible trajectory topologies into distinct trajectory types, based on topological criteria (Figure 4.1c). 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 4.3a.

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.

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 normalised 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.

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 [64]
- **Splatter**: simulations of non-linear paths between different expression states [36]

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 [36]. 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 [35, 65] 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 [66] strongly repress each other, forcing one of the two to become inactive [67]. Such mutual antagonism can be modelled and simulated [68, 69]. 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 [70].

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 [71]), and extracted for every regulator a local network around it using personalised 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 [65]. In this model, the changes in gene expression (x_i) and protein expression (y_i) are modeled using ordinary differential equations [35] (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 normalised such that $\sum_i P_i = 1$.

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

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

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

Similar to GeneNetWeaver [35], 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 [35]. 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 [36] 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 [72, 73], 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 [64], 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 [36] 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 $U(0, 1)$.

Dataset filtering and normalisation

We used a standard single-cell RNA-seq preprocessing pipeline that applies parts of the scran and scater Bioconductor packages [74]. 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.

Benchmark metrics

The importance of using multiple metrics to compare complex models has been stated repeatedly [60]. 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 $F1_{branches}$ compares the branch assignment; (3) the cor_{dist} assesses the similarity in pairwise cell–cell distances and thus the cellular positions; and (4) the $wcor_{features}$ 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 [75] uses the two weighted adjacency matrices of the milestone networks as input (weighted by edge length). It is a linear combination of the normalised Hamming distance, which gives an indication of the differences in edge lengths, and the normalised 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 [58]. 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 $F1_{branches}$ 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 cor_{dist} metric. To calculate the cor_{dist}, 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 cor_{dist} 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 $wcor_{features}$ 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 [76]), 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 normalised the scores on each dataset across the different methods. We shifted and scaled the scores to $\sigma = 1$ and $\mu = 0$, 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.

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.

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.

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-neighbour 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 [39], with $log_{10}|\textit{cells}|$ and $log_{10}|\textit{features}|$ as predictor variables, and fitted this model using the scam function as implemented in the R scam package, with $log_{10}\textit{time}$ (or $log_{10}\textit{memory}$) 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 generalised 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.

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 cor_{dist} and $F1_{branches}$, we only used the intersection between the cells of two datasets, while the intersection of the features was used for the $wcor_{features}$.

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 4.1). 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 [50] 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 4.1). 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 4.1). For calculating the final score, we weighed each of the six categories equally.

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.org), 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).

Reporting Summary

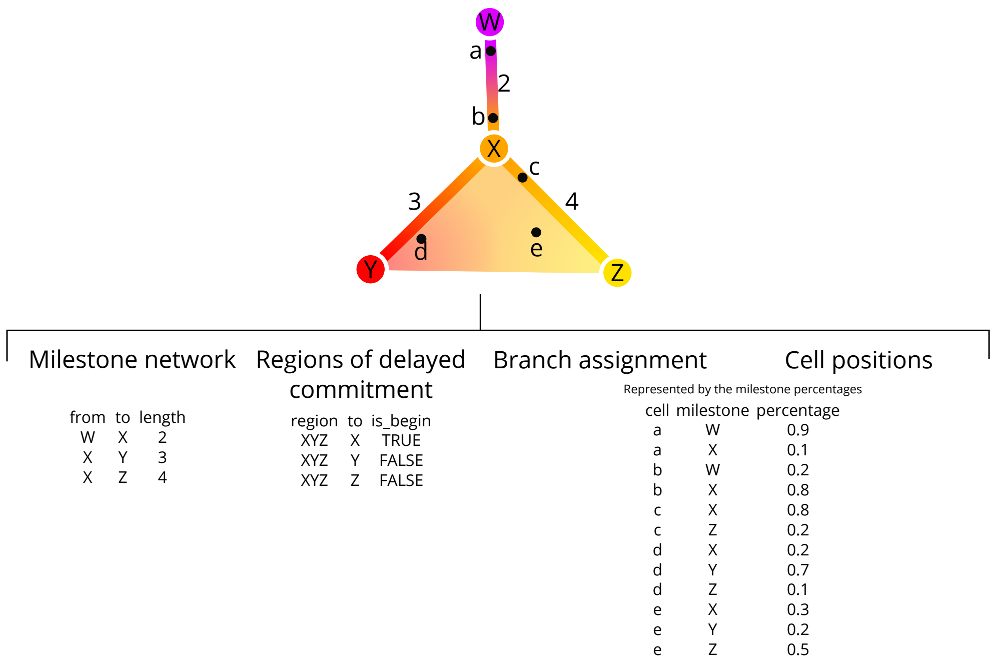
Further information on research design is available in the Nature Research Reporting Summary, available at https://www.nature.com/articles/s41587-019-0071-9#MOESM2

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 4.10).

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


W
a
2
b
X
c
3
4
d
e
Y
Z
Milestone network
Regions of delayed commitment
Branch assignment
Cell positions
Represented by the milestone percentages

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

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

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

Figure 4.10: An example trajectory that will be used throughout this section. It contains contains four milestones (W to Z) and five cells (a to e).

- **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).

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 $\longrightarrow$ B $\longrightarrow$ C $\longrightarrow$ D, C $\longrightarrow$ D, the B milestone was removed
- A linear topology was converted to A $\longrightarrow$ B $\longrightarrow$ C
- A cyclical topology such as A $\longrightarrow$ B $\longrightarrow$ C $\longrightarrow$ D or A $\longrightarrow$ B $\longrightarrow$ A were all simplified to A $\longrightarrow$ B $\longrightarrow$ C $\longrightarrow$ A
- Duplicated edges such as A $\longrightarrow$ B, A $\longrightarrow$ B were decoupled to A $\longrightarrow$ B, A $\longrightarrow$ C $\longrightarrow$ 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 [77], 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 [78]. We implemented a branch and bound approach for this problem, using several heuristics to speed up the search:

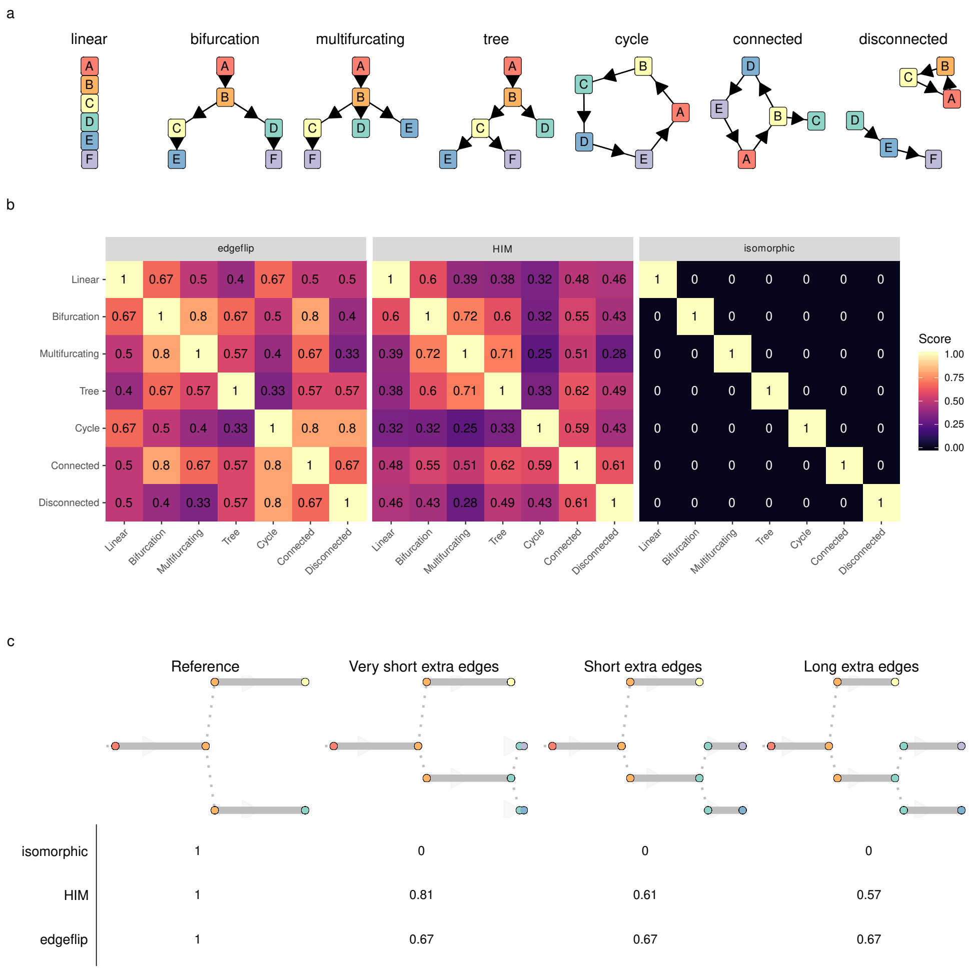
- First check all possible edge additions and removals corresponding to the number of different edges between the two graphs.
- For each possible solution, first check whether:
    1. The maximal degree is the same
    2. The minimal degree is the same
    3. All degrees are the same after sorting
- 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) [75] 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 [79], 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 [80], 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 4.11a). While conceptually very different, the *edgeflip* and *HIM* still produce similar scores (Figure 4.11b). The *HIM* tends to punish the detection of cycles, while the *edgeflip* is more harsh for differences in the number of bifurcations (Figure 4.11b). The main difference however is that the *HIM* takes into account edge lengths when comparing two trajectories, as illustrated in (Figure 4.11c). 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:



edgeflip	Linear	Bifurcation	Multifurcating	Tree	Cycle	Connected	Disconnected
Linear	1	0.67	0.5	0.4	0.67	0.5	0.5
Bifurcation	0.67	1	0.8	0.67	0.5	0.8	0.4
Multifurcating	0.5	0.8	1	0.57	0.4	0.67	0.33
Tree	0.4	0.67	0.57	1	0.33	0.57	0.57
Cycle	0.67	0.5	0.4	0.33	1	0.8	0.8
Connected	0.5	0.8	0.67	0.57	0.8	1	0.67
Disconnected	0.5	0.4	0.33	0.57	0.8	0.67	1

HIM	Linear	Bifurcation	Multifurcating	Tree	Cycle	Connected	Disconnected
Linear	1	0.6	0.39	0.38	0.32	0.48	0.46
Bifurcation	0.6	1	0.72	0.6	0.32	0.55	0.43
Multifurcating	0.39	0.72	1	0.71	0.25	0.51	0.28
Tree	0.38	0.6	0.71	1	0.33	0.62	0.49
Cycle	0.32	0.32	0.25	0.33	1	0.59	0.43
Connected	0.48	0.55	0.51	0.62	0.59	1	0.61
Disconnected	0.46	0.43	0.28	0.49	0.43	0.61	1

isomorphic	Linear	Bifurcation	Multifurcating	Tree	Cycle	Connected	Disconnected
Linear	1	0	0	0	0	0	0
Bifurcation	0	1	0	0	0	0	0
Multifurcating	0	0	1	0	0	0	0
Tree	0	0	0	1	0	0	0
Cycle	0	0	0	0	1	0	0
Connected	0	0	0	0	0	1	0
Disconnected	0	0	0	0	0	0	1

	Reference	Very short extra edges	Short extra edges	Long extra edges
isomorphic	1	0	0	0
HIM	1	0.81	0.61	0.57
edgeflip	1	0.67	0.67	0.67

Figure 4.11: 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 4.12. These clusters of cells can then be compared using one of the many external cluster evaluation measures [58]. 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:

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

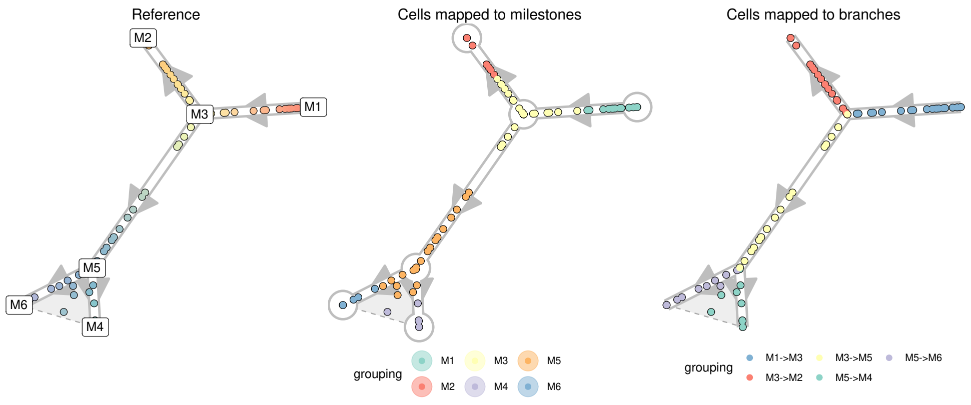


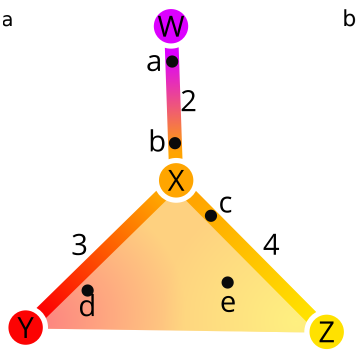
Figure 4.12: Mapping cells to their closest milestone or branch for the calculation of the $F1_{milestones}$ and $F1_{branches}$. To calculate the $F1_{milestones}$, cells are mapped towards the nearest milestone, i.e. the milestone with the highest milestone percentage. For the $F1_{branches}$, the cells are mapped to the closest edge.

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 4.13:

- **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 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

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

d

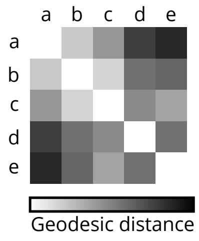


Figure 4.13: The calculation of geodesic distances on a small example trajectory. a) A toy example containing four milestones (W to Z) and five cells (a to e). **b)** The corresponding milestone network, milestone percentages and regions of delayed commitment, when the toy trajectory is converted to the common trajectory model. **c)** The calculations made for calculating the pairwise geodesic distances. **d)** A heatmap representation of the pairwise geodesic distances.

percentages and the length of their shared edge. For cells a and b in the example, $d(a, b)$ is equal to $1 \times (0.9 - 0.2) = 0.7$.

- **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 4.14a). 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 4.14a). We found that using cell waypoints does not introduce 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 4.15)

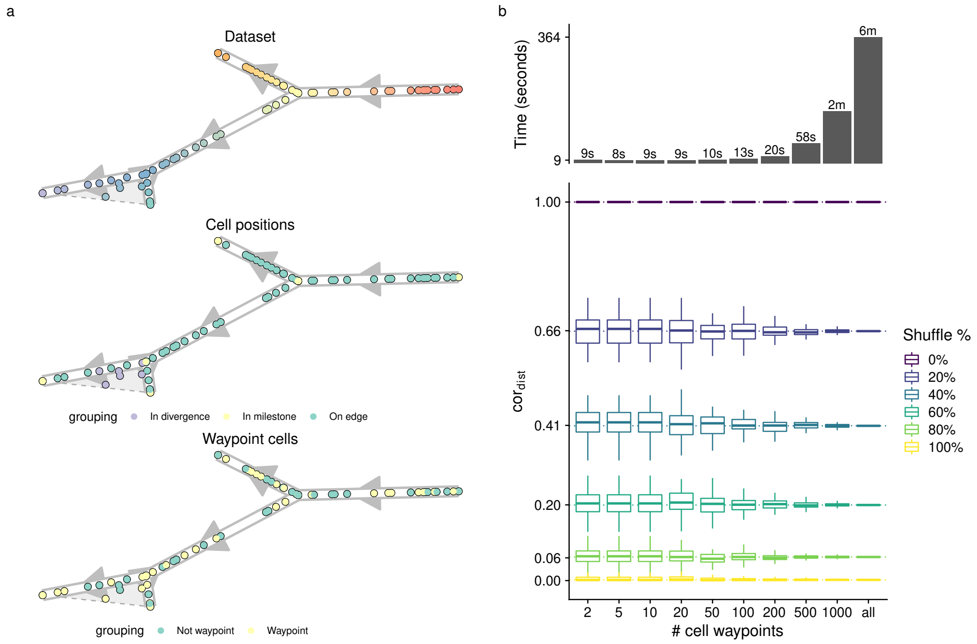


Figure 4.14: Determination of cell waypoints a) Illustration of the stratified cell sampling using an example dataset (top). Each milestone, edge between two milestones and region of delayed commitment is seen as a collection of cells (middle), and the number of waypoints (100 in this case) are divided 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.

with varying topologies. This reduced space captures the original trajectory structure quite well, including the overall topology and branch lengths.

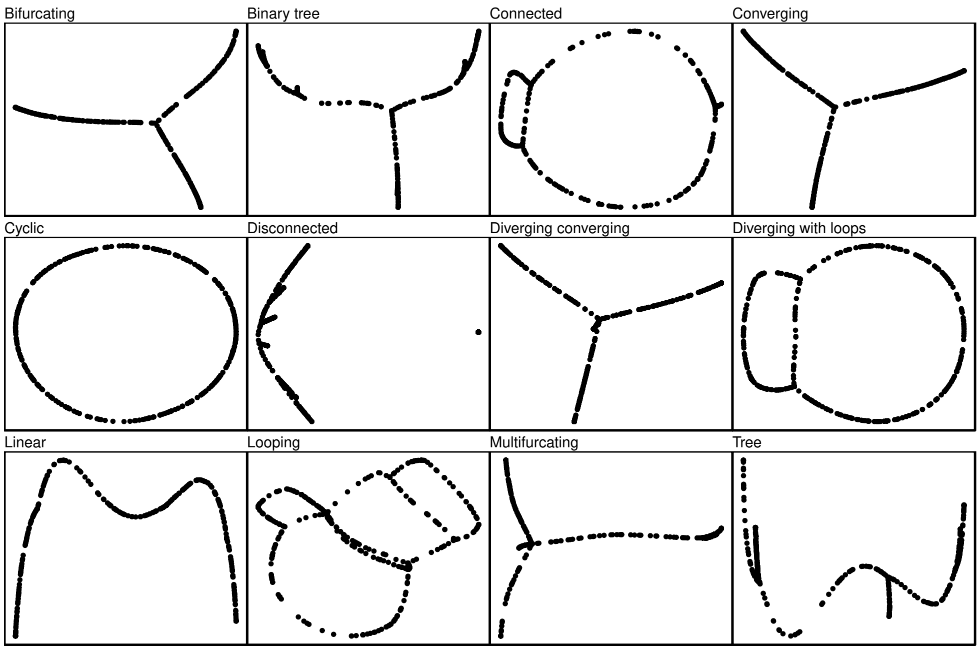
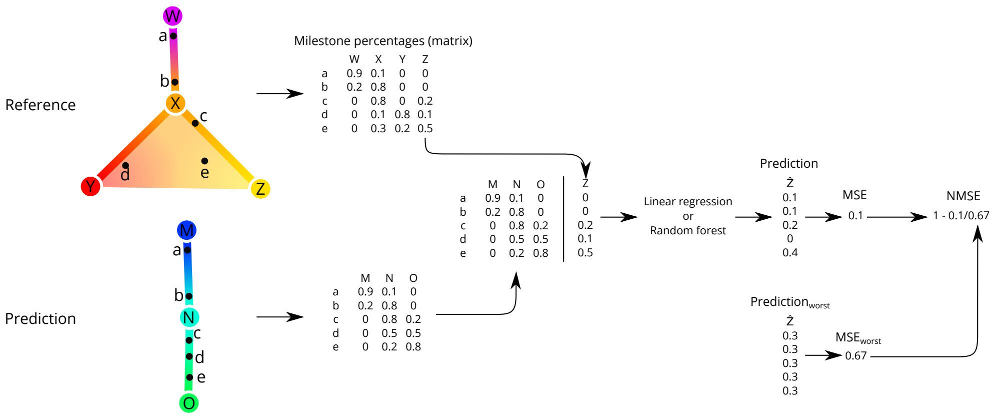


Figure 4.15: Determination of cell waypoints. We generated different toy trajectory datasets with varying topologies and calculated the geodesic distances between all cells within the trajectory. We then used these distances as input for classical multidimensional scaling. This shows that the geodesic distances do not only contain information regarding the cell's positions, but also information on the lengths and wiring of the topology.

$NMSE_{rf}$ 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 4.16). We did this with two regression methods, linear regression (*lm*, using the R lm function) and Random Forest (*rf*, implemented in the *ranger* package [76]). 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 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.



	W	X	Y	Z
a	0.9	0.1	0	0
b	0.2	0.8	0	0
c	0	0.8	0	0.2
d	0	0.1	0.8	0.1
e	0	0.3	0.2	0.5

	M	N	O
a	0.9	0.1	0
b	0.2	0.8	0
c	0	0.8	0.2
d	0	0.5	0.5
e	0	0.2	0.8

	M	N	O	Z
a	0.9	0.1	0	0
b	0.2	0.8	0	0
c	0	0.8	0.2	0.2
d	0	0.5	0.5	0.1
e	0	0.2	0.8	0.5

Ẑ (Prediction)	Ẑ (Prediction$_{worst}$)
0.1	0.3
0.1	0.3
0.2	0.3
0	0.3
0.4	0.3

MSE 0.1; MSE_{worst} 0.67

Figure 4.16: 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.

$cor_{features}$ 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 [76], 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 4.17. 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.

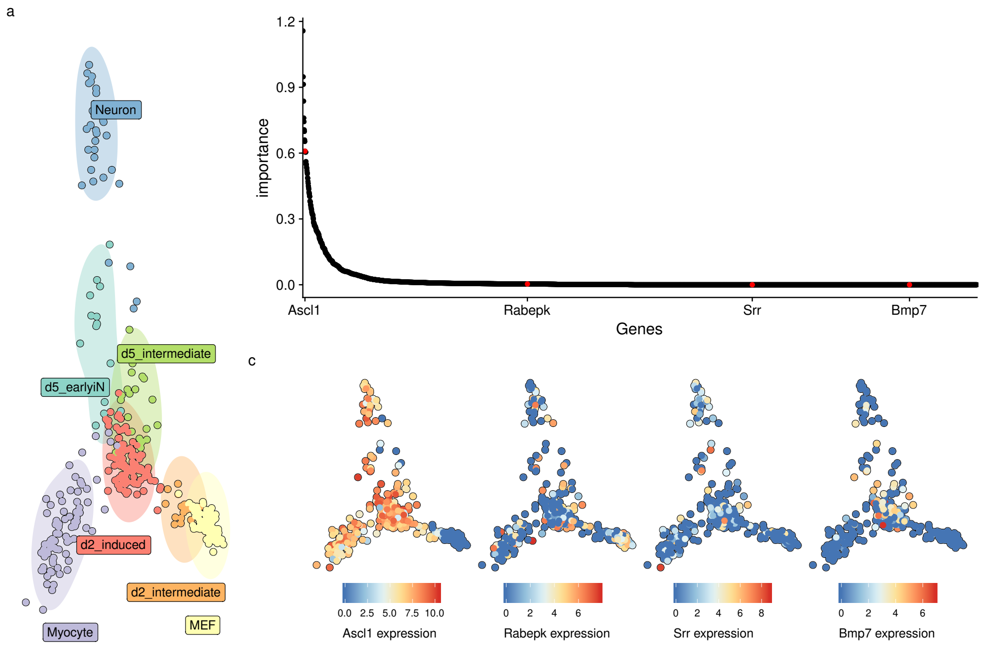


Figure 4.17: An illustration of ranking features based on their importance in a trajectory. (a) A MDS dimensionality reudction of a real dataset in which mouse embryonic fibroblasts (MEF) differentiate into Neurons and Myocytes. (b) The ranking of feature importances from high to low. The majority of 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.

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 4.18. 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.

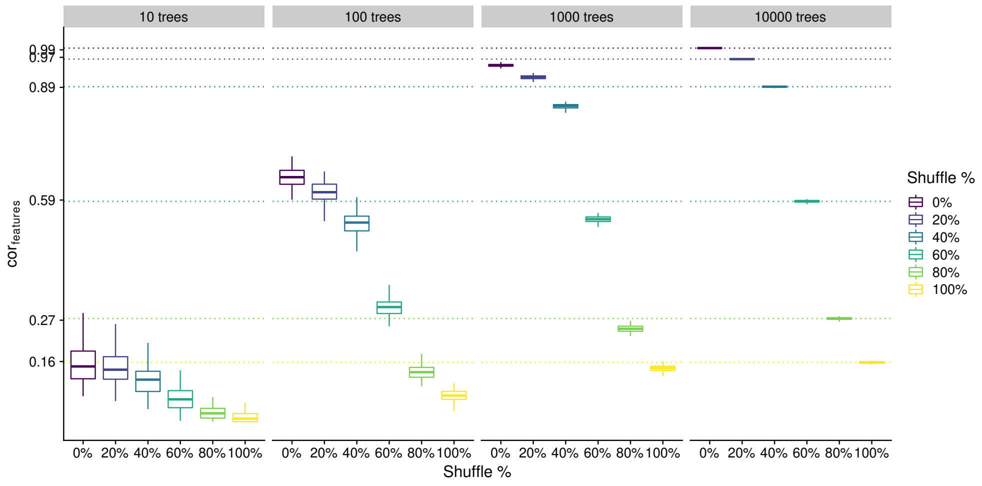


Figure 4.18: Effect of the number of trees parameter on the accuracy and variability of the $cor_{features}$. We used the dataset from Figure 4.17 and calculated the $cor_{features}$ after shuffling a percentage of cells.

For most datasets, only a limited number of features will be differentially expressed in the trajectory. For example, in the dataset used in Figure 4.18 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 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 4.19).

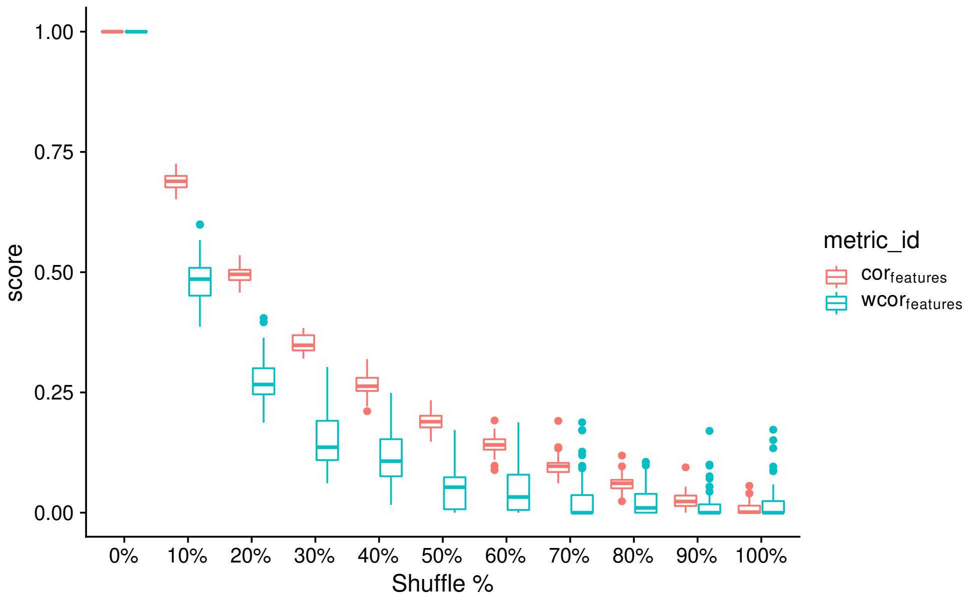


Figure 4.19: Effect of weighting the features based on their feature importance in the reference. We used the same dataset as in Figure 4.17, and calculated the $cor_{features}$ after shuffling a percentage of cells.

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
- # 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 4.2. 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.

Table 4.2: 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	✓	x	✓	✓	✓	✓	✓	x	✓	✓	✓
Local cell shuffling	✓	✓	✓	x	x	x	✓	✓	x	✓	✓
Edge shuffling	✓	✓	✓	x	x	x	✓	✓	✓	✓	✓
Local and global cell shuffling	✓	✓	✓	x	x	x	✓	✓	✓	✓	✓
Changing positions locally and/or globally	✓	✓	✓	x	x	x	✓	✓	x	x	✓
Cell filtering	✓	✓	✓	x	x	x	✓	✓	✓	✓	✓
Removing divergence regions	✓	✓	✓	x	x	x	✓	✓	x	✓	✓
Move cells to start milestone	✓	✓	✓	x	x	x	✓	✓	x	✓	✓
Move cells to closest milestone	✓	✓	✓	x	x	x	✓	✓	x	✓	✓
Length shuffling	✓	x	✓	x	✓	x	x	x	x	✓	✓
Cells into small subedges	x	✓	x	✓	✓	✓	✓	x	✓	✓	✓
New leaf edges	✓	✓	x	✓	✓	✓	x	x	✓	✓	✓
New connecting edges	✓	✓	x	✓	✓	✓	✓	✓	✓	✓	✓
Changing topology and cell position	x	x	x	x	x	x	x	x	x	x	✓
Bifurcation merging	✓	x	✓	✓	✓	✓	✓	✓	✓	✓	✓
Bifurcation merging and changing cell positions	✓	✓	✓	x	x	x	✓	✓	✓	✓	✓
Bifurcation concatentation	✓	x	✓	✓	✓	✓	✓	✓	✓	✓	✓
Cycle breaking	✓	x	✓	✓	✓	✓	✓	✓	x	✓	✓
Linear joining	✓	✓	✓	✓	✓	✓	✓	✓	x	✓	✓
Linear splitting	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Change of topology	✓	x	x	✓	✓	✓	✓	✓	x	✓	✓
Cells on milestones vs edges	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

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.
- **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 4.20. 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 4.20). 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 4.21.

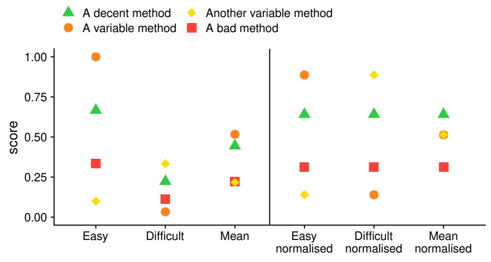
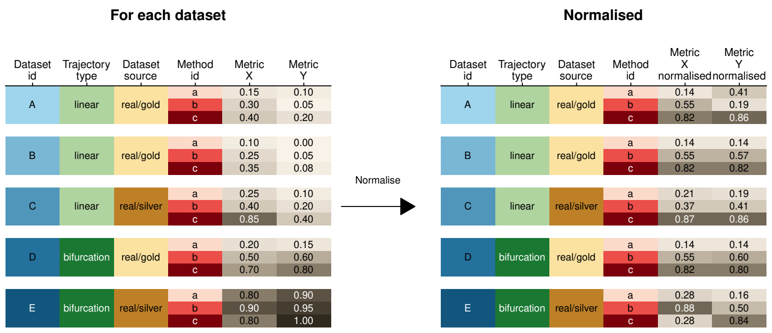


Figure 4.20: An illustration of how the difficulty of a dataset can influence the overall ranking. A decent method, which consistently ranks high on an easy and difficult dataset, does not get a high score when averaging. On the other hand, a method which ranks high on the easy dataset, but very low on the difficult dataset does get a high score on average. After normalising the scores (right), this problem disappears.



For each dataset

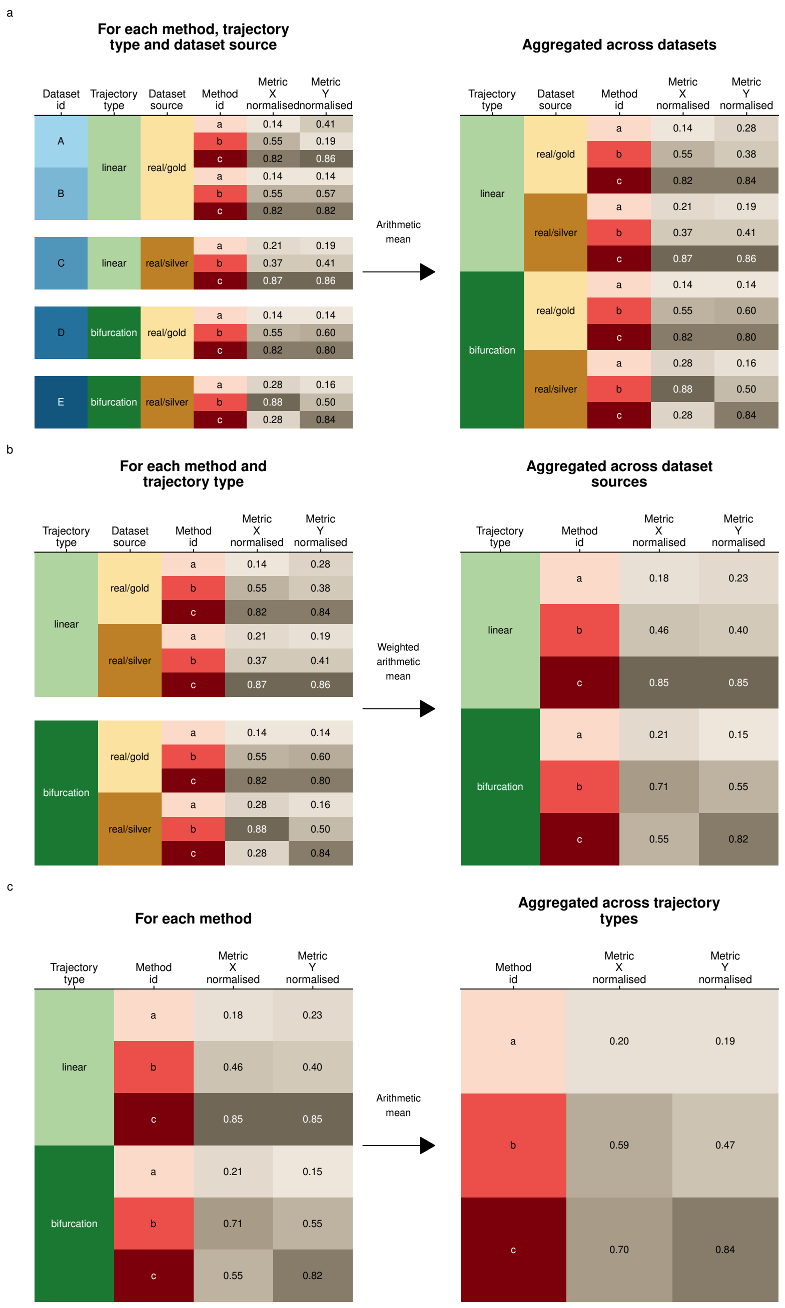
Dataset id	Trajectory type	Dataset source	Method id	Metric X	Metric Y
A	linear	real/gold	a	0.15	0.10
			b	0.30	0.05
			c	0.40	0.20
B	linear	real/gold	a	0.10	0.00
			b	0.25	0.05
			c	0.35	0.08
C	linear	real/silver	a	0.25	0.10
			b	0.40	0.20
			c	0.85	0.40
D	bifurcation	real/gold	a	0.20	0.15
			b	0.50	0.60
			c	0.70	0.80
E	bifurcation	real/silver	a	0.80	0.90
			b	0.90	0.95
			c	0.80	1.00

Normalised

Dataset id	Trajectory type	Dataset source	Method id	Metric X normalised	Metric Y normalised
A	linear	real/gold	a	0.14	0.41
			b	0.55	0.19
			c	0.82	0.86
B	linear	real/gold	a	0.14	0.14
			b	0.55	0.57
			c	0.82	0.82
C	linear	real/silver	a	0.21	0.19
			b	0.37	0.41
			c	0.87	0.86
D	bifurcation	real/gold	a	0.14	0.14
			b	0.55	0.60
			c	0.82	0.80
E	bifurcation	real/silver	a	0.28	0.16
			b	0.88	0.50
			c	0.28	0.84

Figure 4.21: 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 4.22a). 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 4.22b). Finally, the scores are aggregated over the different trajectory types again using a arithmetic mean (Figure 4.22c).


a
For each method, trajectory type and dataset source
Aggregated across datasets
Arithmetic mean
b
For each method and trajectory type
Aggregated across dataset sources
Weighted arithmetic mean
c
For each method
Aggregated across trajectory types
Arithmetic mean

a — For each method, trajectory type and dataset source

Dataset id	Trajectory type	Dataset source	Method id	Metric X normalised	Metric Y normalised
A	linear	real/gold	a	0.14	0.41
A	linear	real/gold	b	0.55	0.19
A	linear	real/gold	c	0.82	0.86
B	linear	real/gold	a	0.14	0.14
B	linear	real/gold	b	0.55	0.57
B	linear	real/gold	c	0.82	0.82
C	linear	real/silver	a	0.21	0.19
C	linear	real/silver	b	0.37	0.41
C	linear	real/silver	c	0.87	0.86
D	bifurcation	real/gold	a	0.14	0.14
D	bifurcation	real/gold	b	0.55	0.60
D	bifurcation	real/gold	c	0.82	0.80
E	bifurcation	real/silver	a	0.28	0.16
E	bifurcation	real/silver	b	0.88	0.50
E	bifurcation	real/silver	c	0.28	0.84

Aggregated across datasets

Trajectory type	Dataset source	Method id	Metric X normalised	Metric Y normalised
linear	real/gold	a	0.14	0.28
linear	real/gold	b	0.55	0.38
linear	real/gold	c	0.82	0.84
linear	real/silver	a	0.21	0.19
linear	real/silver	b	0.37	0.41
linear	real/silver	c	0.87	0.86
bifurcation	real/gold	a	0.14	0.14
bifurcation	real/gold	b	0.55	0.60
bifurcation	real/gold	c	0.82	0.80
bifurcation	real/silver	a	0.28	0.16
bifurcation	real/silver	b	0.88	0.50
bifurcation	real/silver	c	0.28	0.84

b — For each method and trajectory type

Trajectory type	Dataset source	Method id	Metric X normalised	Metric Y normalised
linear	real/gold	a	0.14	0.28
linear	real/gold	b	0.55	0.38
linear	real/gold	c	0.82	0.84
linear	real/silver	a	0.21	0.19
linear	real/silver	b	0.37	0.41
linear	real/silver	c	0.87	0.86
bifurcation	real/gold	a	0.14	0.14
bifurcation	real/gold	b	0.55	0.60
bifurcation	real/gold	c	0.82	0.80
bifurcation	real/silver	a	0.28	0.16
bifurcation	real/silver	b	0.88	0.50
bifurcation	real/silver	c	0.28	0.84

Aggregated across dataset sources

Trajectory type	Method id	Metric X normalised	Metric Y normalised
linear	a	0.18	0.23
linear	b	0.46	0.40
linear	c	0.85	0.85
bifurcation	a	0.21	0.15
bifurcation	b	0.71	0.55
bifurcation	c	0.55	0.82

c — For each method

Trajectory type	Method id	Metric X normalised	Metric Y normalised
linear	a	0.18	0.23
linear	b	0.46	0.40
linear	c	0.85	0.85
bifurcation	a	0.21	0.15
bifurcation	b	0.71	0.55
bifurcation	c	0.55	0.82

Aggregated across trajectory types

Method id	Metric X normalised	Metric Y normalised
a	0.20	0.19
b	0.59	0.47
c	0.70	0.84

Figure 4.22: 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.

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:

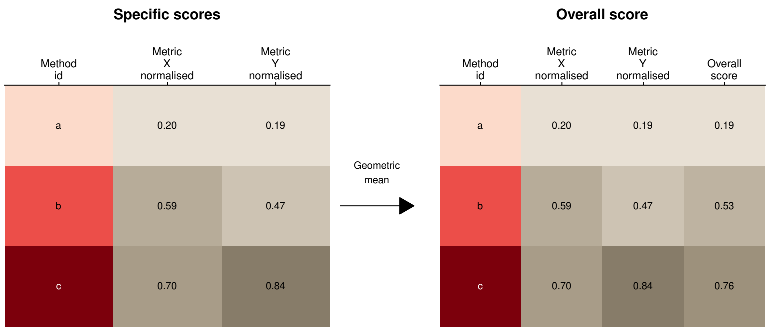
- Its **ordering**, using the cor_{dist}
- Its **branch assignment**, using the $F1_{branches}$
- Its **topology**, using the *HIM*
- The accuracy of **differentially expressed features**, using the $wcor_{features}$

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 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 4.23) 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


Specific scores
Overall score
Geometric mean

Method id	Metric X normalised	Metric Y normalised
a	0.20	0.19
b	0.59	0.47
c	0.70	0.84

Method id	Metric X normalised	Metric Y normalised	Overall score
a	0.20	0.19	0.19
b	0.59	0.47	0.53
c	0.70	0.84	0.76

Figure 4.23: An example of the averaging procedure. For each method, we calculated the geometric mean between its normalised and aggregated scores

score to order the methods. Such a context-dependent ranking of all methods is provided through the dynguidelines app (http://guidelines.dynverse.org).

Update

Our study has already had some impact on the field. Several studies use our method wrappers [81], use our synthetic data generator [82], or were inspired by our figure design [83, 82, 84].

Is this the final answer to which trajectory method is best? Not at all. There are still many challenges left to resolve, scalability being only one of them. As my colleagues at the institute found out multiple times: even if the trajectory is obvious when looking at a dimensionality reduction, that does not mean that the trajectory is rapidly detected by the method. The TI field might follow a similar course as biclustering: an initial burst, followed by a slow but steady continuous improvement of the methods. RNA velocity emerged right after we were finishing the benchmark study [59], and might be a powerful tool to better define the trajectories in the data.

One of the goals of the benchmark was to support development of new, better tools. We will therefore soon organise a competition in collaboration with the Laboratory for Innovation Science at Harvard. The competition will reuse parts of the existing benchmark, but updated in some aspects to reflect recent developments. I'm really excited about this opportunity, because it could bring a completely new set of ideas within the trajectory inference field.

This study really convinced me of the power of using preprints. It made it possible for us to disseminate our results almost as soon as we were confident with them. Moreover, because several method users were already aware of our study, they were more inclined to answer and address our feedback on their method. For us, publishing the preprint also provided some time to improve the paper and code. This allowed us to migrate each method wrapper inside dockers, add a guidelines app, and improve the metrics [85].

At the same time, it also highlights some of the ongoing challenges of using preprints. Several of the methods that we evaluated were only published in preprint, and were as of May 2019 never published in a peer-reviewed journal. It might be possible that methods which performed lower in our benchmark had more difficulties with getting published, because they were already deemed obsolete. In a world where science is progressing ever faster thanks to the internet, might it become even harder to establish yourself in a quickly evolving field?

References

[1] Amos Tanay and Aviv Regev. "Scaling Single-Cell Genomics from Phenomenology to Mechanism". In: *Nature* 541.7637 (Jan. 18, 2017), nature21350. ISSN: 1476-4687. DOI: 10.1038/nature21350.

[2] Martin Etzrodt, Max Endele, and Timm Schroeder. "Quantitative Single-Cell Approaches to Stem Cell Research". In: *Cell Stem Cell* 15.5 (Nov. 6, 2014), pp. 546–558. ISSN: 1934-5909. DOI: 10.1016/j.stem.2014.10.015.

[3] Cole Trapnell. "Defining Cell Types and States with Single-Cell Genomics". In: *Genome Research* 25.10 (Jan. 10, 2015), pp. 1491–1498. ISSN: 1088-9051, 1549-5469. DOI: 10.1101/gr.190595.115.

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

[5] Kevin R. Moon et al. "Manifold Learning-Based Methods for Analyzing Single-Cell RNA-Sequencing Data". In: *Current Opinion in Systems Biology.* • Future of Systems Biology• Genomics and Epigenomics 7 (Feb. 1, 2018), pp. 36–46. ISSN: 2452-3100. DOI: 10.1016/j.coisb.2017.12.008.

[6] Zehua Liu et al. "Reconstructing Cell Cycle Pseudo Time-Series via Single-Cell Transcriptome Data". In: *Nature Communications* 8.1 (June 19, 2017), p. 22. ISSN: 2041-1723. DOI: 10.1038/s41467-017-00039-z.

[7] F. Alexander Wolf et al. "Graph Abstraction Reconciles Clustering with Trajectory Inference through a Topology Preserving Map of Single Cells". In: *bioRxiv* (Oct. 25, 2017), p. 208819. DOI: 10.1101/208819.

[8] 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.

[9] 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.

[10] Peter See et al. "Mapping the Human DC Lineage through the Integration of High-Dimensional Techniques". In: *Science* 356.6342 (June 9, 2017), eaag3009. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.aag3009.

[11] Sara Aibar et al. "SCENIC: Single-Cell Regulatory Network Inference and Clustering". In: *Nature Methods* 14.11 (Nov. 2017), pp. 1083–1086. ISSN: 1548-7105. DOI: 10.1038/nmeth.4463.

[12] Aviv Regev et al. "Science Forum: The Human Cell Atlas". In: *eLife* 6 (Dec. 5, 2017), e27041. ISSN: 2050-084X. DOI: 10.7554/eLife.27041.

[13] Xiaoping Han et al. "Mapping the Mouse Cell Atlas by Microwell-Seq". In: *Cell* 172.5 (Feb. 22, 2018), 1091–1107.e17. ISSN: 1097-4172. DOI: 10.1016/j.cell.2018.02.001.

[14] Nicholas Schaum et al. "Single-Cell Transcriptomics of 20 Mouse Organs Creates a Tabula Muris". In: *Nature* 562.7727 (Oct. 1, 2018), pp. 367–372. ISSN: 1476-4687. DOI: 10.1038/s41586-018-0590-4.

[15] 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 • Pharmacology and Drug Discovery 4 (Aug. 1, 2017), pp. 85–91. ISSN: 2452-3100. DOI: 10.1016/j.coisb.2017.07.004.

[16] 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 14, 2014). ISSN: 1758-0463. DOI: 10.1093/database/bau069.

[17] Sean Davis et al. *https://github.com/seandavi/awesome-single-cell*. Zenodo, June 20, 2018. DOI: 10.5281/zenodo.1294021.

[18] Luke Zappia, Belinda Phipson, and Alicia Oshlack. "Exploring the Single-Cell RNA-Seq Analysis Landscape with the scRNA-Tools Database". In: *bioRxiv* (Oct. 20, 2017), p. 206573. DOI: 10.1101/206573.

[19] Sean C. Bendall et al. "Single-Cell Trajectory Detection Uncovers Progression and Regulatory Coordination in Human B Cell Development". In: *Cell* 157.3 (Apr. 24, 2014), pp. 714–725. ISSN: 0092-8674. DOI: 10.1016/j.cell.2014.04.005.

[20] Jaehoon Shin et al. "Single-Cell RNA-Seq with Waterfall Reveals Molecular Cascades Underlying Adult Neurogenesis". In: *Cell Stem Cell* 17.3 (Sept. 3, 2015), pp. 360–372. ISSN: 1875-9777. DOI: 10.1016/j.stem.2015.07.013.

[21] Kieran Campbell and Christopher Yau. "Bayesian Gaussian Process Latent Variable Models for Pseudotime Inference in Single-Cell RNA-Seq Data". In: *bioRxiv* (Sept. 15, 2015), p. 026872. DOI: 10.1101/026872.

[22] 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.

[23] Manu Setty et al. "Wishbone Identifies Bifurcating Developmental Trajectories from Single-Cell Data". In: *Nature Biotechnology* 34.6 (June 2016), pp. 637–645. ISSN: 1546-1696. DOI: 10.1038/nbt.3569.

[24] 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. 23, 2014), nbt.2859. ISSN: 1546-1696. DOI: 10.1038/nbt.2859.

[25] Hirotaka Matsumoto and Hisanori Kiryu. "SCOUP: A Probabilistic Model Based on the Ornstein–Uhlenbeck Process to Analyze Single-Cell Expression Data during Differentiation". In: *BMC Bioinformatics* 17 (June 8, 2016), p. 232. ISSN: 1471-2105. DOI: 10.1186/s12859-016-1109-3.

[26] 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.

[27] Kelly Street et al. "Slingshot: Cell Lineage and Pseudotime Inference for Single-Cell Transcriptomics". In: *BMC Genomics* 19.1 (June 19, 2018), p. 477. ISSN: 1471-2164. DOI: 10.1186/s12864-018-4772-0.

[28] Zhicheng Ji and Hongkai Ji. "TSCAN: Pseudo-Time Reconstruction and Evaluation in Single-Cell RNA-Seq Analysis". In: *Nucleic Acids Research* 44.13 (July 27, 2016), e117–e117. ISSN: 0305-1048. DOI: 10.1093/nar/gkw430.

[29] 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 (May 23, 2016), p. 106. ISSN: 1474-760X. DOI: 10.1186/s13059-016-0975-3.

[30] 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. 13, 2016), p. 363. ISSN: 1471-2105. DOI: 10.1186/s12859-016-1175-6.

[31] Robrecht Cannoodt et al. "SCORPIUS Improves Trajectory Inference and Identifies Novel Modules in Dendritic Cell Development". In: *bioRxiv* (Oct. 7, 2016), p. 079509. DOI: 10.1101/079509.

[32] 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. 3, 2017), eaal2192. ISSN: 2470-9468. DOI: 10.1126/sciimmunol.aal2192.

[33] 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. 15, 2017), p. 19. ISSN: 2398-502X. DOI: 10.12688/wellcomeopenres.11087.1.

[34] Luyi Tian et al. "scRNA-Seq Mixology: Towards Better Benchmarking of Single Cell RNA-Seq Protocols and Analysis Methods". In: *bioRxiv* (Oct. 3, 2018), p. 433102. DOI: 10.1101/433102.

[35] Thomas Schaffter, Daniel Marbach, and Dario Floreano. “GeneNetWeaver: In Silico Benchmark Generation and Performance Profiling of Network Inference Methods”. In: *Bioinformatics (Oxford, England)* 27.16 (Aug. 15, 2011), pp. 2263–2270. ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btr373.

[36] Luke Zappia, Belinda Phipson, and Alicia Oshlack. “Splatter: Simulation of Single-Cell RNA Sequencing Data”. In: *Genome Biology* 18 (Sept. 12, 2017), p. 174. ISSN: 1474-760X. DOI: 10.1186/s13059-017-1305-0.

[37] 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.

[38] Junyue Cao et al. “Joint Profiling of Chromatin Accessibility and Gene Expression in Thousands of Single Cells”. In: *Science* (Aug. 30, 2018), eaau0730. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.aau0730.

[39] Natalya Pya and Simon N. Wood. “Shape Constrained Additive Models”. In: *Statistics and Computing* 25.3 (May 1, 2015), pp. 543–559. ISSN: 1573-1375. DOI: 10.1007/s11222-013-9448-7.

[40] Morgan Taschuk and Greg Wilson. “Ten Simple Rules for Making Research Software More Robust”. In: *PLOS Computational Biology* 13.4 (Apr. 13, 2017), e1005412. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1005412.

[41] Serghei Mangul et al. “A Comprehensive Analysis of the Usability and Archival Stability of Omics Computational Tools and Resources”. In: *bioRxiv* (Oct. 25, 2018), p. 452532. DOI: 10.1101/452532.

[42] Greg Wilson et al. “Best Practices for Scientific Computing”. In: *PLOS Biology* 12.1 (Jan. 7, 2014), e1001745. ISSN: 1545-7885. DOI: 10.1371/journal.pbio.1001745.

[43] Haydee Artaza et al. “Top 10 Metrics for Life Science Software Good Practices”. In: *F1000Research* 5 (Aug. 16, 2016), p. 2000. ISSN: 2046-1402. DOI: 10.12688/f1000research.9206.1.

[44] Jeff Lee. *Rpackages: R Package Development - the Leek Group Way!* Dec. 27, 2017.

[45] Hadley Wickham. *R Packages: Organize, Test, Document, and Share Your Code.* ”O’Reilly Media, Inc.”, Mar. 26, 2015. 201 pp. ISBN: 978-1-4919-1056-6.

[46] Luis Bastiao Silva et al. “General Guidelines for Biomedical Software Development”. In: *F1000Research* 6 (July 12, 2017). ISSN: 2046-1402. DOI: 10.12688/f1000research.10750.2.

[47] Rafael C. Jiménez et al. “Four Simple Recommendations to Encourage Best Practices in Research Software”. In: *F1000Research* 6 (June 13, 2017). ISSN: 2046-1402. DOI: 10.12688/f1000research.11407.1.

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

[49] Alex Anderson. *Writing Great Scientific Code.* Oct. 12, 2016.

[50] Brett K. Beaulieu-Jones and Casey S. Greene. “Reproducibility of Computational Workflows Is Automated Using Continuous Analysis”. In: *Nature Biotechnology* 35.4 (Mar. 13, 2017), nbt.3780. ISSN: 1546-1696. DOI: 10.1038/nbt.3780.

[51] Vincent Driessen. *A Successful Git Branching Model.* Jan. 5, 2010. URL: http://nvie.com/posts/a-successful-git-branching-model/ (visited on 03/28/2018).

[52] Anne-Laure Boulesteix. “Ten Simple Rules for Reducing Overoptimistic Reporting in Methodological Computational Research”. In: *PLOS Computational Biology* 11.4 (Apr. 23, 2015), e1004191. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1004191.

[53] Jean Francois Puget. *Green Dice Are Loaded (Welcome to p-Hacking).* Mar. 22, 2016.

[54] Frank Gannon. “The Essential Role of Peer Review”. In: *EMBO Reports* 2.9 (Sept. 15, 2001), p. 743. ISSN: 1469-221X. DOI: 10.1093/embo-reports/kve188.

[55] Melinda Baldwin. “In Referees We Trust?” In: *Physics Today* 70.2 (Feb. 1, 2017), pp. 44–49. ISSN: 0031-9228. DOI: 10.1063/PT.3.3463.

[56] 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.

[57] Monika Jelizarow et al. "Over-Optimism in Bioinformatics: An Illustration". In: *Bioinformatics* 26.16 (Aug. 15, 2010), pp. 1990–1998. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btq323.

[58] Wouter Saelens, Robrecht Cannoodt, and Yvan Saeys. "A Comprehensive Evaluation of Module Detection Methods for Gene Expression Data". In: *Nature Communications* 9.1 (Mar. 15, 2018), p. 1090. ISSN: 2041-1723. DOI: 10.1038/s41467-018-03424-4.

[59] 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.

[60] 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 (Jan. 1, 2011), p. 537. ISSN: 1744-4292, 1744-4292. DOI: 10.1038/msb.2011.70.

[61] Anthony Gitter. *https://github.com/agitter/single-cell-pseudotime*. June 25, 2018. DOI: 10.5281/zenodo.1297423.

[62] Tsukasa Kouno et al. "Temporal Dynamics and Transcriptional Control Using Single-Cell Gene Expression Analysis". In: *Genome Biology* 14 (Dec. 10, 2013), R118. ISSN: 1474-760X. DOI: 10.1186/gb-2013-14-10-r118.

[63] 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.

[64] Nikolaos Papadopoulos, Rodrigo Gonzalo Parra, and Johannes Soeding. "PROSSTT: Probabilistic Simulation of Single-Cell RNA-Seq Data for Complex Differentiation Processes". In: *bioRxiv* (Jan. 31, 2018), p. 256941. DOI: 10.1101/256941.

[65] Daniel Marbach et al. "Wisdom of Crowds for Robust Gene Network Inference". In: *Nature methods* 9.8 (July 15, 2012), pp. 796–804. ISSN: 1548-7091. DOI: 10.1038/nmeth.2016.

[66] Heping Xu et al. "Regulation of Bifurcating B Cell Trajectories by Mutual Antagonism between Transcription Factors IRF4 and IRF8". In: *Nature Immunology* 16.12 (Dec. 2015), pp. 1274–1281. ISSN: 1529-2916. DOI: 10.1038/ni.3287.

[67] 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.

[68] 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 17, 2011), pp. 8257–8262. ISSN: 0027-8424, 1091-6490. DOI: 10.1073/pnas.1017017108.

[69] James E. Ferrell. "Bistability, Bifurcations, and Waddington's Epigenetic Landscape". In: *Current Biology* 22.11 (June 5, 2012), R458–R466. ISSN: 0960-9822. DOI: 10.1016/j.cub.2012.03.045.

[70] Nir Yosef et al. "Dynamic Regulatory Network Controlling TH17 Cell Differentiation". In: *Nature* 496.7446 (Apr. 25, 2013), pp. 461–468. ISSN: 1476-4687. DOI: 10.1038/nature11981.

[71] Daniel Marbach et al. "Tissue-Specific Regulatory Circuits Reveal Variable Modular Perturbations across Complex Diseases". In: *Nature Methods* 13.4 (Apr. 2016), pp. 366–370. ISSN: 1548-7105. DOI: 10.1038/nmeth.3799.

[72] Toni Giorgino. "Computing and Visualizing Dynamic Time Warping Alignments in R: The Dtw Package". In: *Journal of Statistical Software* 31.7 (Sept. 30, 2009). DOI: 10.18637/jss.v031.i07.

[73] 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. 1, 2009), pp. 11–34. ISSN: 0933-3657. DOI: 10.1016/j.artmed.2008.11.007.

[74] 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. 31, 2016), p. 2122. ISSN: 2046-1402. DOI: 10.12688/f1000research.9501.2.

[75] 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)*.

2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA). Oct. 2015, pp. 1–10. DOI: 10.1109/DSAA.2015.7344816.

[76] 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. 31, 2017). DOI: 10.18637/jss.v077.i01.

[77] 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)*. 0 vols. Proceedings. Society for Industrial and Applied Mathematics, Jan. 6, 2007, pp. 135–149. DOI: 10.1137/1.9781611972870.13.

[78] 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 — Gramado, Brazil, 2009 160.18 (Dec. 1, 2012), pp. 2523–2541. ISSN: 0166-218X. DOI: 10.1016/j.dam.2012.01.026.

[79] 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.

[80] 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. 046109. ISSN: 1539-3755. DOI: 10.1103/PhysRevE.66.046109.

[81] Xiuwei Zhang, Chenling Xu, and Nir Yosef. “Simulating Multiple Faceted Variability in Single Cell RNA Sequencing”. In: *Nature Communications* 10.1 (June 13, 2019), p. 2611. ISSN: 2041-1723. DOI: 10.1038/s41467-019-10500-w.

[82] Aditya Pratapa et al. “Benchmarking Algorithms for Gene Regulatory Network Inference from Single-Cell Transcriptomic Data”. In: *bioRxiv* (June 4, 2019), p. 642926. DOI: 10.1101/642926.

[83] Shiquan Sun et al. “Accuracy, Robustness and Scalability of Dimensionality Reduction Methods for Single Cell RNAseq Analysis”. In: *bioRxiv* (May 17, 2019), p. 641142. DOI: 10.1101/641142.

[84] Saskia Freytag et al. “Comparison of Clustering Tools in R for Medium-Sized 10x Genomics Single-Cell RNA-Sequencing Data”. In: *F1000Research* 7 (Dec. 19, 2018), p. 1297. ISSN: 2046-1402. DOI: 10.12688/f1000research.15809.2.

[85] Wouter Saelens et al. “A Comparison of Single-Cell Trajectory Inference Methods: Towards More Accurate and Robust Tools”. In: *bioRxiv* (Mar. 5, 2018), p. 276907. DOI: 10.1101/276907.

5 Conclusion

In this thesis, I highlighted my three main contributions to the analysis of transcriptomics data. I first described a tool to analyse the transcriptome and functional diversity of three biological conditions. I then presented our benchmark of module detection methods, where I highlight several guidelines for users and developers. I concluded with another benchmark, this time on methods that can infer trajectories from single-cell transcriptomics data. Here too we end with several guidelines, along with a set of tools that can be reused to run and evaluate trajectory inference methods. These tools are briefly highlighted in a future perspective chapter (Chapter 6), where I highlight some ongoing work, and my vision on where the field should (or will) move.

The main topic of this thesis - transcriptomics - changed considerably throughout this thesis. At first a technology to assess the transcriptional effect of some perturbation on a sample or cell population, it has now become a technology to profile a whole tissue or organism in an unbiased way. And as can be observed throughout the thesis, more 'samples' mean that new powerful analyses, such as trajectory inference, become a possibility. These in turn need to be increasingly scalable to handle the increasingly large datasets.

As a cheap and broadly applicable technology, transcriptomics is here to stay. Furthermore, many recent technological advances are broadening its usefulness as we speak. These include long-read sequencing, spatial transcriptomics and combination with other omics technologies. As these techniques mature, so will the analysis of transcriptomics data have to evolve. This will require new, accurate, scalable, user-friendly and robust tools. As we have seen, this often results in an explosion of many tools within a couple of years, highlighting the need for thorough benchmarks. These can be used for developers to explore new algorithmic avenues and compare them with the state of the art. Users on the other hand get a clear picture of what tools are available, what they can do, and how they are expected to perform on their use case.

Overall, both benchmarking studies presented in this thesis reveal the importance of such objective comparisons for the community. In both studies, the best performing methods were often not the most popular ones. In this sense, benchmarking may make these methods more approachable by users, speeding up their analysis while making it more powerful. On the other hand, benchmarks also show that good ideas or novel algorithms do not necessarily lead to good performance. In both benchmarks, quite simple methods were often the top performing. In the module detection case, these concerned methods such as independent component analysis, which have existed long before transcriptomics data analysis. For trajectory inference, methods that perform clustering together with some basic preprocessing, such as PAGA and Slingshot, outperform those with complex Bayesian models. This indicates that developers of methods should avoid overcomplicating things, and should instead focus more on making their method scalable, usable and robust.

As benchmarks presented in this thesis and elsewhere show, some methods do perform worse than others. One may wonder though: how is this possible, if most of these methods were developed by competent scientists and checked through peer-review? The studies in this thesis provide some possible reaons for this. For one, performance can be very variable across datasets, while most methods are only optimised during development on a couple of datasets. Datasets also change as the field develops, and some aspects such as scalability may only become important after a methods publication. Finally, the quality of the implementation also has a large impact on the benchmarking score. Good documentation allows the benchmarker to write a good wrapper for the method, and sane error checking allows it to be run on most if not all datasets. These criteria are not met for a substantial number of methods, with a lower performance as a consequence.

Nonetheless, the usefulness of benchmarking is still relatively limited for both target audiences. Especially in a fast moving field such as single-cell transcriptomics, new methods are published every month. This renders the benchmark paper rapidly obsolete, which brings us back to the question: ”Which tool to use, the old state-of-the-art or the new tools?”. For developers, the usefulness of the benchmark hinges on how easy it is to compare a new tool with the state-of-the-art. But what we observe is that benchmarks are often (partly) re-implemented. This is often not the fault of the developer, because new tools typically include new functionality that was not tested by older benchmarks. In both cases, this makes it clear that benchmarks have to evolve together with the tools that are developed. I propose a plan on how to do this in another future perspectives chapter (Chapter 7).

As a final critical note: is benchmarking really necessary? Wouldn't you expect a researcher to easily spot wrong models made by a method, e.g. through visualisation? It's true that in some cases, an inaccurate model may simply cause a minor annoyance and a waste of time/money, without any long-term effects on the study. In this case, a benchmarking study may seem excessive. But at the other hand of the spectrum, wrong models or hypotheses may lead to long-term futile search to confirm the hypothesis, or a fruitless development of new drugs that are bound to be ineffective. The importance of a benchmark thus depends on what is done with a resulting model. If it is just seen as a hypothesis, that is still checked through independent methods, then benchmarks may not be so useful. If the model is seen as evidence, then it is critical that the validity of the underlying methods is checked thoroughly.

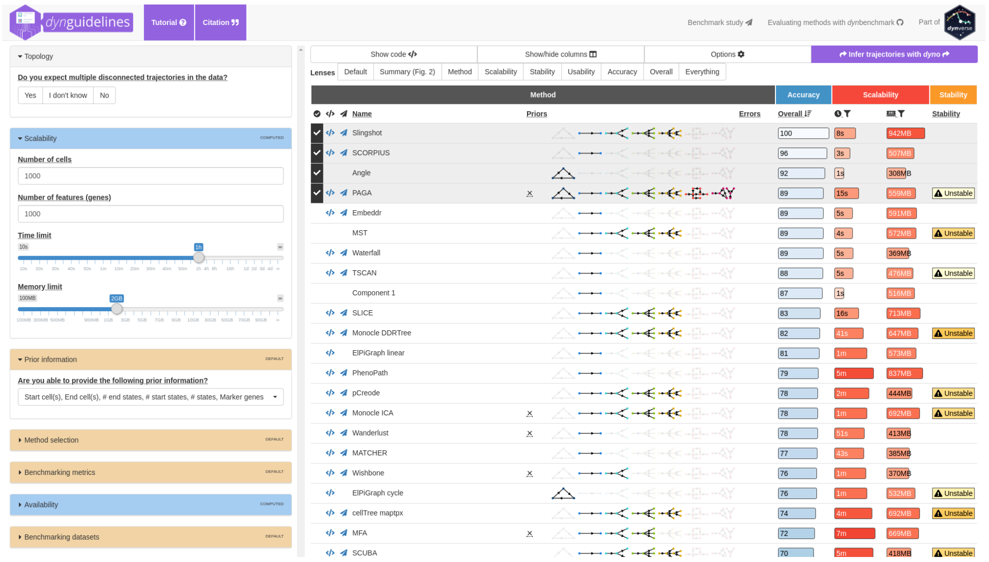
6 | Future perspectives - The next milestones of trajectory inference

While it may not immediately visible when reading the paper, a proper benchmarking study involves the development of a lot of auxiliary code to make all parts fit together. For the trajectory inference benchmark, we created several R packages that preprocess data, wrap methods, visualise their output, and compare different output models. These are all functionalities that would also be very useful for method users and developers. That's why we're now integrating these packages into a streamlined toolkit (called *dyno*) to infer trajectories from single-cell data.

Despite limited advertisement, this toolkit is already being used by several researchers according to activity on our github (https://github.com/dynverse/dyno). The basic functionality of *dyn*o is presented in the vignette in Appendix B. However, I still have many ideas on how to improve the toolkit further, which are highlighted in this future perspectives chapter. The further we get into the chapter, the more I discuss long-term plans, which probably also necessitate technological improvements to go along with the computational ones.

Selecting the most optimal set of methods

The first step within the toolkit is to select a set of methods that need to be run, for which we provide an interactive app that was developed within the context of the TI benchmarking study (http://guidelines.dynverse.org/). The app provides users with several questions regarding the expected topology in the data, the dimensions of the data, the computational resources a user has to their disposal, and the prior information that may be present (Figure 6.1). Based on the answers to these questions, we provide context-specific guidelines. The most optimal set of methods is selected based on their accuracy on the datasets with particular topologies, and based on whether the user has sufficient computational resources. A user is warned if a method often fails to run, or when its output may be unstable. In this way, the guidelines app makes sure that a user always use the most appropriate methods for their use case, and prepares the user for unwanted failure, stability and scalability issues.



Topology

Do you expect multiple disconnected trajectories in the data?

Yes | I don't know | No

Scalability COMPUTED

Number of cells: 1000

Number of features (genes): 1000

Time limit: 10s – 1h – ∞

Memory limit: 100MB – 2GB – ∞

Prior information DEFAULT

Are you able to provide the following prior information?

Start cell(s), End cell(s), # end states, # start states, # states, Marker genes

Method selection DEFAULT

Benchmarking metrics DEFAULT

Availability COMPUTED

Benchmarking datasets DEFAULT

Method			Accuracy	Scalability		Stability
	Name	Priors	Overall			Stability
✓	Slingshot		100	8s	942MB	
✓	SCORPIUS		96	3s	507MB	
✓	Angle		92	1s	308MB	
✓	PAGA	×	89	15s	559MB	Unstable
	Embeddr		89	5s	591MB	
	MST		89	4s	572MB	Unstable
	Waterfall		89	5s	369MB	
	TSCAN		88	5s	476MB	Unstable
	Component 1		87	1s	516MB	
	SLICE		83	16s	713MB	
	Monocle DDRTree		82	41s	647MB	Unstable
	ElPiGraph linear		81	1m	573MB	
	PhenoPath		79	5m	837MB	
	pCreode		78	2m	444MB	Unstable
	Monocle ICA	×	78	1m	692MB	Unstable
	Wanderlust	×	78	51s	413MB	
	MATCHER		77	43s	385MB	
	Wishbone	×	76	1m	370MB	
	ElPiGraph cycle		76	1m	532MB	Unstable
	cellTree maptpx		74	4m	692MB	Unstable
	MFA	×	72	7m	669MB	
	SCUBA		70	5m	418MB	Unstable

Figure 6.1: A screenshot of guidelines.dynverse.org. Depending on answers to questions (left), the selection of methods is changed (right). The user also gets extra information for each method, such as expected running time, whether it errors often, and whether its results are unstable.

The app was developed in a modular way, such that the content (the questions, answers and benchmarking results) is strictly separated from the back-end (the sorting and filtering of methods based on questions). This makes it possible to reuse the app for other benchmarking studies, such as evaluations of clustering methods [1] (including the one presented in this thesis) or normalisation methods [2]. This still requires some trivial separation of the back-end into a separate package, with its own clear tutorials, documentation and publication. There are also opportunities here to make the app more interactive, for example by allowing users to post feedback about a particular tool.

Running any method without difficulties

To run one method, a user typically has to spend more than half a day with installation, format conversion and debugging. In some cases, this is further complicated because of dependency conflicts or system-dependent installation problems. To circumvent this within our benchmark, we installed each method within a container. These containers also contain a wrapper script that makes it possible to call a method from the command line, and run each method within a common input and output interface. Running many methods thus becomes as simple as changing one parameter in the code (as long as the container environment is installed):

```
trajectory <- infer_trajectory(dataset, "slingshot")
```

to

```
trajectory <- infer_trajectory(dataset, "paga")
```

Containerisation also makes it easier for developers to add a method within our framework, provided they make use of the same input and output formats. They are no longer restricted to particular programming languages, and are free to do anything inside the container environment. This may however be a curse as well, as it could promote bad code practices, and may even allow malicious code to be executed if the container is ran without proper input/output control. Although containers were therefore of tremendous value within our evaluation, it may not be the ideal solution for bio-informatics tools in general (and biomedical bio-informatics in particular). Perhaps an intermediate solution such as conda [3], or creating organisations that manage containers (such as bioboxes or biocontainers [4]), may be necessary.

Adapting and post-processing the trajectory

A trajectory is only a model of the data, and often needs some post-processing to make it biologically interpretable. This can involve annotating particular milestones, and orienting the edges of the trajectory in a biologically meaningful way. At the moment, our toolkit contains manual methods to adapt the trajectory. This may involve labelling based on the expression of a certain marker, or orienting a trajectory based on a given start cell. However, ideally these steps should be automated as much as possible, so that the analysis becomes more objective.

With large-scale cell atlases from different organises on the horizon, automated cell annotation of single-cell data is becoming realistic. This is exemplified by the many annotation tools that have recently been published [1] [5, 6, 7]. Cell annotation tools often look for enrichment of certain markers within a cell cluster, and how this translates in a trajectory setting still has to be investigated. Annotation of milestones at the leaves of the trajectory resemble the typical use case quite well. But what if some cell types were not assigned a milestone by the TI method, should an annotation tool then ”create” an intermediate milestone? Further complicating this is that distinct cellular states may be difficult to define during complex differentiations such as hematopoiesis [8]. Whether annotation tools can be useful for trajectories should therefore still be investigated. The best way to tackle this may be to start slowly, with datasets where the differentiation stages are well known and validated, such as cerebral cortex development [9] or gut organoids [10].

Most TI methods have no idea about the orientation of cell differentiation. Although there is often a biologically plausibile orientation, it can still be useful to have this confirmed in a data-driven way. The best way to do this may be by orienting the trajectory with RNA velocity vectors of nearby cells. I have done some initial tests on this (Figure 6.2) which look promising. But it has also highlighted some challenges. RNA velocity vectors can be very noisy and parameter-dependent when only limited number of cells are present. Moreover, what if the velocity vector switches its orientation in the middle of a trajectory edge, should the tool then ”create” an intermediate milestone? Some datasets, for example those coming from plants, have a very low number of intronic reads and here alternative approaches will also be necessary. An integration with lineage tracing [11] may be necessary in such cases.

Visualising the trajectory

Visualisation is of utmost importance when interpreting and sharing research. Within our toolkit, a minimal set of plotting is already implemented, which make it possible to recreate plots such as Figure 4.8. However, from the many comments that we received at conferences or through Github, we have found that our plotting library is insufficient for many current use cases. There are countless ways in which single-cell data (and trajectory data in particular) can be plotted, four of which are presented in Figure 6.2. Creating these plots using monolithic functions (such as the ones in our package, but also in popular interpretation packages such as Seurat [13] and scanpy [14]) results in either an overly complex or an overly restricted function.

Instead, our idea is to design a plotting library that heavily uses ideas from the ”grammar of graphics” [15], as implemented in the ggplot2 package [16]. This is a more modular system to describe visualisation, in which data points are mapped to

[1] You might say that annotation tools are now in a ”boom” phase, just like trajectory inference in 2017-2018

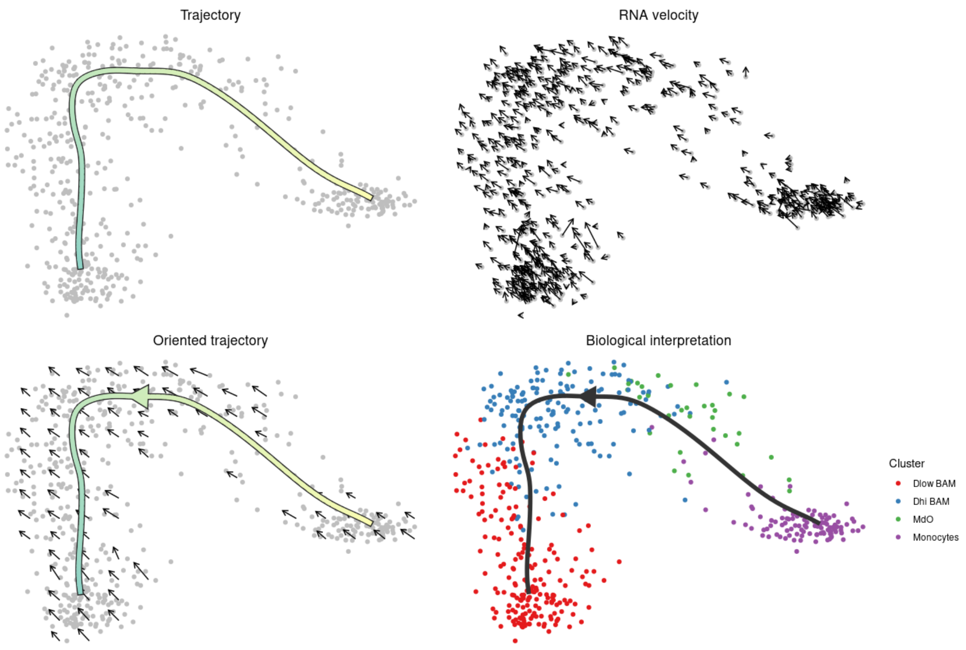


Figure 6.2: Different visualisations of the same trajectory on a dataset of mouse brain macrophages [12]: dural MHC-II low macrophages (Dlow BAM), dural MHC-II high macrophages (Dhigh BAM), monocyte derived cells (MdO) and monocytes.

certain geometrical objects: circles, points along a line, arrows, ... Parts of these geometrical objects can vary depending on the data: the position may depend on the dimensionality reduction, the colour may depend on the annotation, ... In this way, a plot is built from the bottom up using separate elements. As exemplified by ggplot2, it makes plotting also more easy to extend. This might prove useful in the future when new data types such as the spatial location [17] and RNA acceleration [18] come available.

Comparing trajectories

One trajectory describes how a cell changes, but multiple trajectories can tell you something about how the trajectory itself changes. Comparing similar trajectories between patients, genotypes or after perturbation might in the long run be the most useful application of TI methods. This would provide some mechanistic insights into how these factors impact cellular dynamics, and could give some ideas about how to influence these trajectories to cure or improve differentiation. One might imagine many ways in which trajectories could be different (Figure 6.3); the flux towards particular branches may change, there may be a blockade at some point, the expression of a branch may shift at some points, or a combination of any of these. Better understanding such trajectory changes could be relevant to better understand diseases such as primary immunodeficiencies [19], or could improve the efficiency of cellular reprogramming [20].

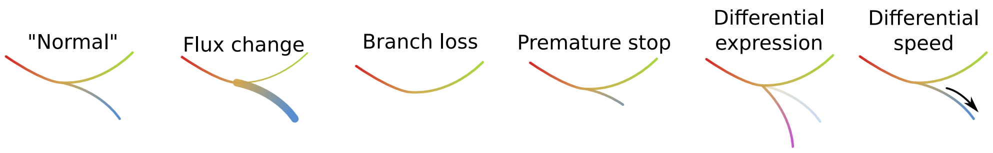


Figure 6.3: Different possible scenarios that can change a cellular trajectory. Compared to the ”normal” trajectory, the flux towards particular branches may change (flux change), cell may no longer choose for a particular branch (branch loss), a branch may stop prematurely (premature stop), the expression in a branch may change (differential expression) or the speed by which the cells progress may change (differential speed).

Before one can compare trajectories, they first need to be aligned. This is straightforward for linear trajectories, where the problem boils down to classical linear alignment, with many proven dynamic programming techniques [21]. Aligning trees or more complex trajectories might prove a bit more tricky, given the combinatorial complexity by which trajectories could be aligned. An alternative way might be to integrate the alignment within the TI methods itself, as has already been done for linear trajectories [22]. This is an approach that is often used for clustering populations, although appropriate batch correction [23] or cell hashing [24, 25] techniques may be necessary.

After alignment, statistics will be necessary to define what is actually different between the trajectories [26]. It might be useful to use concepts from clustering here, for example those that try to find both differential abundance and differential expression [27].

Multi-omics trajectories

TI methods have also been applied on other data types than transcriptomics, including ATAC-seq [28, 29] and cytometry [30]. Single-cell multi-omics techniques are now becoming mainstream, measuring both the transcriptome with protein expression [31, 32] or chromatin accessibility [33] of the same set of cells. This provides new opportunities for studying the dynamics of cells, for example to delineate the earliest stages at which a cell makes a fate decision. Simply using existing methods on the concatenated dataset might not be the best approach, given the differences in distribution and relationships between individual features. A better alternative is probably to infer trajectories on separate data types, and then integrate it through alignment as described earlier [34]. An even more powerful approach might be to first integrate all data types [35, 36], and apply existing or new TI methods on this joint representation.

Validating a trajectory

All TI methods have, as far as we know, a common fundamental problem: they find a trajectory even if the data does not support this. Some methods, such as ouija [37] do provide some cell-wise uncertainty measures, but these values cannot be used to validate the full trajectory without a reference. Currently, a quality check therefore occurs by a manual validation of the trajectory, for example through visualisation. Clustering has a similar problem, and there it is alleviated by using either internal quality measures or by including external information. How these would translate to trajectory models is still unclear.

An internal measure for TI methods might be whether the trajectory always follows the most dense parts of the dataset. Alternatively, one might look at the stability of the trajectory [38], as is often done in clustering [39]. The tricky part for these measures is the definition of a ”null” value and ”positive” value. The more the actual value lies towards the positive value, the better the trajectory is supported. These reference values might be estimated from datasets that are known to contain one or no trajectories [40].

The most promising way to externally validate a trajectory might be the inclusion of RNA velocity information. A measure could for example quantify whether the RNA velocity vector of each cell is perpendicular with nearby trajectory edges.

In the end, a trajectory remains one of many possible models of the data. Actual proof of cell differentiation should be sought by experimental means, such as lineage tracing or live cell imaging. The use case of these methods is typically still limited to *in vitro* or primary tissue samples, although some data types might be useful to do tracing *in vivo* [41]. Promising in this regard are combinations of the two approaches [11], although it has to be seen how easy such technologies can be deployed *in vivo*.

Conclusion

In the summer of 2016, I wrote in our review on TI methods that ”[...] current studies have only explored the tip of the iceberg of what TI can offer.” Three years later, and with more than 60 new methods available, this still applies. There are clear opportunities for using trajectories to study or improve biological systems. But some technological (batch effects) and computational (lack of user friendly tools and powerful algorithms) limitations still hold us back.

At the technological side, batch effects make it difficult to distinguish relevant variation from irrelevant biological or technical variation. Cell hashing techniques [24], or computational alignment techniques [42, 23], may mitigate this issue.

At the computational side, there is a clear lack of tools that make it easier to explore, visualise and interpret trajectories. A lot of effort has been done in developing new methods, but only a handful of tools also provide the essentials for interpretation [28, 43]. Moreover, new methods will have to be developed to cope with more complex use cases, such as trajectory comparisons and multi-omics trajectories. It might be useful to borrow some concepts here from (single-cell) clustering.

The goal with our toolkit is to unite these different tools into a streamlined pipeline, with as little data conversion as possible. Our current toolkit (https://github.com/dynverse/dyno) already provides the essentials, but there is still a lot of work to be done (and people to be convinced) to make this work. One approach that really excites me is to use the toolkit within a crowdsourcing framework, so that state-of-the-art tools are immediately available for potential users. We are currently working together with the Laboratory for Innovation Science at Harvard to make this happen (https://www.topcoder.com/challenges/30092303).

References

[1] Angelo Duò, Mark D. Robinson, and Charlotte Soneson. "A Systematic Performance Evaluation of Clustering Methods for Single-Cell RNA-Seq Data". In: *F1000Research* 7 (Sept. 10, 2018), p. 1141. ISSN: 2046-1402. DOI: 10.12688/f1000research.15666.2.

[2] Beate Vieth et al. "A Systematic Evaluation of Single Cell RNA-Seq Analysis Pipelines: Library Preparation and Normalisation Methods Have the Biggest Impact on the Performance of scRNA-Seq Studies". In: *bioRxiv* (Mar. 19, 2019), p. 583013. DOI: 10.1101/583013.

[3] Björn Grüning et al. "Bioconda: Sustainable and Comprehensive Software Distribution for the Life Sciences". In: *Nature Methods* 15.7 (July 2018), p. 475. ISSN: 1548-7105. DOI: 10.1038/s41592-018-0046-7.

[4] Felipe da Veiga Leprevost et al. "BioContainers: An Open-Source and Community-Driven Framework for Software Standardization". In: *Bioinformatics* 33.16 (Aug. 15, 2017), pp. 2580–2582. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btx192.

[5] Genevieve L. Stein-O'Brien et al. "Decomposing Cell Identity for Transfer Learning across Cellular Measurements, Platforms, Tissues, and Species". In: *Cell Systems* 8.5 (May 22, 2019), 395–411.e8. ISSN: 2405-4720. DOI: 10.1016/j.cels.2019.04.004.

[6] Chenling Xu et al. "Harmonization and Annotation of Single-Cell Transcriptomics Data with Deep Generative Models". In: *bioRxiv* (Jan. 29, 2019), p. 532895. DOI: 10.1101/532895.

[7] Dvir Aran et al. "Reference-Based Analysis of Lung Single-Cell Sequencing Reveals a Transitional Profibrotic Macrophage". In: *Nature Immunology* 20.2 (Feb. 2019), p. 163. ISSN: 1529-2916. DOI: 10.1038/s41590-018-0276-y.

[8] Amir Giladi et al. "Single-Cell Characterization of Haematopoietic Progenitors and Their Trajectories in Homeostasis and Perturbed Haematopoiesis". In: *Nature Cell Biology* 20.7 (July 2018), p. 836. ISSN: 1476-4679. DOI: 10.1038/s41556-018-0121-4.

[9] Silvia Velasco et al. "Individual Brain Organoids Reproducibly Form Cell Diversity of the Human Cerebral Cortex". In: *Nature* (June 5, 2019), p. 1. ISSN: 1476-4687. DOI: 10.1038/s41586-019-1289-x.

[10] Adam L. Haber et al. "A Single-Cell Survey of the Small Intestinal Epithelium". In: *Nature* 551.7680 (Nov. 16, 2017), pp. 333–339. ISSN: 0028-0836. DOI: 10.1038/nature24489.

[11] Lennart Kester and Alexander van Oudenaarden. "Single-Cell Transcriptomics Meets Lineage Tracing". In: *Cell Stem Cell* 23.2 (Aug. 2, 2018), pp. 166–179. ISSN: 1934-5909. DOI: 10.1016/j.stem.2018.04.014.

[12] Hannah Van Hove et al. "A Single-Cell Atlas of Mouse Brain Macrophages Reveals Unique Transcriptional Identities Shaped by Ontogeny and Tissue Environment". In: *Nature Neuroscience* 22.6 (June 2019), p. 1021. ISSN: 1546-1726. DOI: 10.1038/s41593-019-0393-4.

[13] Andrew Butler et al. "Integrating Single-Cell Transcriptomic Data across Different Conditions, Technologies, and Species". In: *Nature Biotechnology* 36.5 (May 2018), pp. 411–420. ISSN: 1546-1696. DOI: 10.1038/nbt.4096.

[14] F. Alexander Wolf, Philipp Angerer, and Fabian J. Theis. "SCANPY: Large-Scale Single-Cell Gene Expression Data Analysis". In: *Genome Biology* 19.1 (Feb. 6, 2018), p. 15. ISSN: 1474-760X. DOI: 10.1186/s13059-017-1382-0.

[15] Leland Wilkinson. *The Grammar of Graphics*. Springer Science & Business Media, Jan. 28, 2006. 693 pp. ISBN: 978-0-387-28695-2.

[16] Hadley Wickham. *Ggplot2: Elegant Graphics for Data Analysis*. Use R! New York: Springer-Verlag, 2009. ISBN: 978-0-387-98141-3.

[17] Samuel G. Rodriques et al. "Slide-Seq: A Scalable Technology for Measuring Genome-Wide Expression at High Spatial Resolution". In: *Science* 363.6434 (Mar. 29, 2019), pp. 1463–1467. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.aaw1219.

[18] Gennady Gorin, Valentine Svensson, and Lior Pachter. "RNA Velocity and Protein Acceleration from Single-Cell Multiomics Experiments". In: *bioRxiv* (June 6, 2019), p. 658401. DOI: 10.1101/658401.

[19] Nikita Raje and Chitra Dinakar. "Overview of Immunodeficiency Disorders". In: *Immunology and allergy clinics of North America* 35.4 (Nov. 2015), pp. 599–623. ISSN: 0889-8561. DOI: 10.1016/j.iac.2015.07.001.

[20] Geoffrey Schiebinger et al. "Optimal-Transport Analysis of Single-Cell Gene Expression Identifies Developmental Trajectories in Reprogramming". In: *Cell* 176.4 (Feb. 7, 2019), 928–943.e22. ISSN: 0092-8674. DOI: 10.1016/j.cell.2019.01.006.

[21] Ayelet Alpert et al. "Alignment of Single-Cell Trajectories to Compare Cellular Expression Dynamics". In: *Nature Methods* 15.4 (Apr. 2018), pp. 267–270. ISSN: 1548-7105. DOI: 10.1038/nmeth.4628.

[22] Kieran R. Campbell and Christopher Yau. "Uncovering Pseudotemporal Trajectories with Covariates from Single Cell and Bulk Expression Data". In: *Nature Communications* 9.1 (June 22, 2018), p. 2442. ISSN: 2041-1723. DOI: 10.1038/s41467-018-04696-6.

[23] Tim Stuart et al. "Comprehensive Integration of Single-Cell Data". In: *Cell* 0.0 (June 6, 2019). ISSN: 0092-8674, 1097-4172. DOI: 10.1016/j.cell.2019.05.031.

[24] Marlon Stoeckius et al. "Cell Hashing with Barcoded Antibodies Enables Multiplexing and Doublet Detection for Single Cell Genomics". In: *Genome Biology* 19.1 (Dec. 19, 2018), p. 224. ISSN: 1474-760X. DOI: 10.1186/s13059-018-1603-1.

[25] Christopher S. McGinnis et al. "MULTI-Seq: Sample Multiplexing for Single-Cell RNA Sequencing Using Lipid-Tagged Indices". In: *Nature Methods* (June 17, 2019), p. 1. ISSN: 1548-7105. DOI: 10.1038/s41592-019-0433-8.

[26] Koen Van den Berge et al. "Trajectory-Based Differential Expression Analysis for Single-Cell Sequencing Data". In: *bioRxiv* (May 2, 2019), p. 623397. DOI: 10.1101/623397.

[27] Lukas M. Weber et al. "Diffcyt: Differential Discovery in High-Dimensional Cytometry via High-Resolution Clustering". In: *Communications Biology* 2.1 (May 14, 2019), p. 183. ISSN: 2399-3642. DOI: 10.1038/s42003-019-0415-5.

[28] Huidong Chen et al. "Single-Cell Trajectories Reconstruction, Exploration and Mapping of Omics Data with STREAM". In: *Nature Communications* 10.1 (Apr. 23, 2019), p. 1903. ISSN: 2041-1723. DOI: 10.1038/s41467-019-09670-4.

[29] Hannah A. Pliner et al. "Cicero Predicts Cis-Regulatory DNA Interactions from Single-Cell Chromatin Accessibility Data". In: *Molecular Cell* 71.5 (June 9, 2018), 858–871.e8. ISSN: 1097-4164. DOI: 10.1016/j.molcel.2018.06.044.

[30] Manu Setty et al. "Wishbone Identifies Bifurcating Developmental Trajectories from Single-Cell Data". In: *Nature Biotechnology* 34.6 (June 2016), pp. 637–645. ISSN: 1546-1696. DOI: 10.1038/nbt.3569.

[31] Vanessa M. Peterson et al. "Multiplexed Quantification of Proteins and Transcripts in Single Cells". In: *Nature Biotechnology* 35.10 (Oct. 2017), pp. 936–939. ISSN: 1546-1696. DOI: 10.1038/nbt.3973.

[32] Marlon Stoeckius et al. "Simultaneous Epitope and Transcriptome Measurement in Single Cells". In: *Nature Methods* 14.9 (Sept. 2017), pp. 865–868. ISSN: 1548-7105. DOI: 10.1038/nmeth.4380.

[33] Longqi Liu et al. "Deconvolution of Single-Cell Multi-Omics Layers Reveals Regulatory Heterogeneity". In: *Nature Communications* 10.1 (Jan. 28, 2019), p. 470. ISSN: 2041-1723. DOI: 10.1038/s41467-018-08205-7.

[34] Joshua D. Welch, Alexander J. Hartemink, and Jan F. Prins. "MATCHER: Manifold Alignment Reveals Correspondence between Single Cell Transcriptome and Epigenome Dynamics". In: *Genome Biology* 18.1 (July 24, 2017), p. 138. ISSN: 1474-760X. DOI: 10.1186/s13059-017-1269-0.

[35] Ricard Argelaguet et al. “Multi-Omics Factor Analysis—a Framework for Unsupervised Integration of Multi-omics Data Sets”. In: *Molecular Systems Biology* 14.6 (June 1, 2018), e8124. ISSN: 1744-4292, 1744-4292. DOI: 10.15252/msb.20178124.

[36] M. Colomé-Tatché and F. J. Theis. “Statistical Single Cell Multi-Omics Integration”. In: *Current Opinion in Systems Biology.* • Future of Systems Biology• Genomics and Epigenomics 7 (Feb. 1, 2018), pp. 54–59. ISSN: 2452-3100. DOI: 10.1016/j.coisb.2018.01.003.

[37] Kieran R. Campbell and Christopher Yau. “A Descriptive Marker Gene Approach to Single-Cell Pseudotime Inference”. In: *Bioinformatics (Oxford, England)* 35.1 (Jan. 1, 2019), pp. 28–35. ISSN: 1367-4811. DOI: 10.1093/bioinformatics/bty498.

[38] Robrecht Cannoodt et al. “SCORPIUS Improves Trajectory Inference and Identifies Novel Modules in Dendritic Cell Development”. In: *bioRxiv* (Oct. 7, 2016), p. 079509. DOI: 10.1101/079509.

[39] Vladimir Yu Kiselev et al. “SC3: Consensus Clustering of Single-Cell RNA-Seq Data”. In: *Nature Methods* 14.5 (May 2017), pp. 483–486. ISSN: 1548-7105. DOI: 10.1038/nmeth.4236.

[40] Luyi Tian et al. “Benchmarking Single Cell RNA-Sequencing Analysis Pipelines Using Mixture Control Experiments”. In: *Nature Methods* 16.6 (June 2019), p. 479. ISSN: 1548-7105. DOI: 10.1038/s41592-019-0425-8.

[41] Jin Xu et al. “Single-Cell Lineage Tracing by Endogenous Mutations Enriched in Transposase Accessible Mitochondrial DNA”. In: *eLife* 8 (Apr. 9, 2019). ISSN: 2050-084X. DOI: 10.7554/eLife.45105.

[42] Ilya Korsunsky et al. “Fast, Sensitive, and Accurate Integration of Single Cell Data with Harmony”. In: *bioRxiv* (Nov. 5, 2018), p. 461954. DOI: 10.1101/461954.

[43] 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.

7 | Future perspectives - A roadmap for continuous and collaborative benchmarking

In this future perspectives chapter, I describe a workflow for making benchmarking more continuous and collaborative. This was developed from my experience of being involved in two benchmarking studies, and from intensive discussions with bioinformatics researchers from Ghent and Zürich. In brief, the workflow uses several tools available from modern software development and applies them for designing a benchmarking study. The actual implementation of this workflow will still require some time to make all components fit well together, and to write detailed documentation for potential participants.

Introduction

Evaluating the performance of a new method, and comparing it to the state-of-the-art, is a critical step in the development of bioinformatics methods. The breadth of a benchmark is influenced by its purpose. In some studies, the goal is to review the methods available in the field, and highlight current challenges. Such independent benchmarks are usually very comprehensive, involving many datasets and different metrics ranging assessing the accuracy, scalability and robustness of a method. A special case of such a benchmark are competitions, where the focus lies on promoting the development of new methods within the field, while using existing methods as baseline. Other benchmarks are used as a companion to a study proposing a new method, which aim at demonstrating its improvements and usefulness.

While benchmarks are clearly important, the way benchmarking is usually done has some limitations:

- Benchmarks are quickly outdated when new methods come along. This is especially problematic in fast moving fields, such as single-cell transcriptomics; our TI benchmarking study is now already outdated given some recently published methods [1]. Benchmarks also need to evolve with the methods that are being published; a benchmark of TI methods in 2016 would have evaluated only on linear and bifurcating datasets, while methodological developments now require the evaluation of tree and disconnected graphs as well.
- Benchmarks are difficult to extend, as this is usually only added as an afterthought. The use of containerisation and code availability already partially alleviates this issue.
- Benchmarks often reach different conclusions and these are difficult to compare, because of (unclear) differences in datasets, method parameters, metric implementation and aggregation.
- Independent benchmarks and competitions tend to be authoritative, with only a small group of people deciding on how methods should be compared.
- Independent benchmarks are usually published quite late, only after a lot of methods are already available.
- Companion benchmarks represent considerable wasted effort, because datasets are often reanalysed, metrics reimplemented, and methods rewrapped.

To resolve these issues, we propose a workflow for benchmarking that centers around the following three core concepts:

- **Modular**: It should be possible to extend the benchmark simply by adding a self-contained "module". Such a module could be: a dataset generator, a method, a set of metrics, or a report generator that interprets the results and produces a report. Several tools exist already for making benchmarks modular: SummarizedBenchmark [2], Dynamic Statistical Comparisons (https://github.com/stephens999/dscr) and iCOBRA [3].
- **Collaborative**: Anyone with a computer and an internet connection should be able to run and contribute to the benchmark. This can range from contributing a module, to changing the structure of the benchmark itself. Discussions on the benchmark or any of the reports should also be open. The collaborative aspect of benchmarking has usually focused on the level of methods, with countless competitions and challenges, such as those organised by DREAM

(http://dreamchallenges.org/), kaggle (https://www.kaggle.com/) or topcoder (https://www.topcoder.com/).

- **Continuous**: A benchmark should be continuously updated when new modules are added. This has quite a long history in bioinformatics, particularly in structure prediction [4], but also in other fields such as transcriptomics [5].

To construct a workflow that fulfills these three concepts, we consolidated several ideas and tools coming from modern software development, such as continuous integration, containerisation and workflow management.

In brief, our workflow is structured as follows. We define several **types of modules** (Figure 7.1a): dataset generators that can generate datasets and optionally use another dataset as input; methods that use a dataset to generate some model; metrics which calculate some scores using the model and optionally also parts of the dataset; and finally a report generator that summarise the datasets, models and scores. Each type of module can generate a set of files which are constrained to a particular set of **formats** (Figure 7.1b). Each format has an unambiguous description, example data, and includes a validator that verifies the output files generated by each module. While each format is defined beforehand, new formats can be added over time as the field progresses. A **module** (Figure 7.1c) is a set of scripts and packages, which are run inside a portable environment. This module is put under version control, shared on a code sharing platform, and tested automatically. When all tests of a module are successful, these modules can be integrated into the actual **benchmarking workflow** (Figure 7.1d). Within this workflow, modules are connected through a particular design, which is executed using a workflow manager. The output of the benchmark are a set of reports and apps, which are made available through a publishing platform. To add a new module, a pull request is created to integrate the module within the benchmarking workflow, after which the contribution is reviewed openly. When accepted, the module is automatically integrated within the workflow, and the necessary parts of the workflow are re-executed. Finally, in regular time intervals (e.g. monthly), the full set of reports and apps are gathered and versioned.

We will further discuss each element of the workflow in detail, with some possible tools that can be used to implement it. It is important to acknowledge that this is only one possible implementation, and that other tools, some of which still have to be developed, could better fit the benchmarking use-case. What is the most important is not the way our workflow is implemented, but the open philosophy behind its implementation.

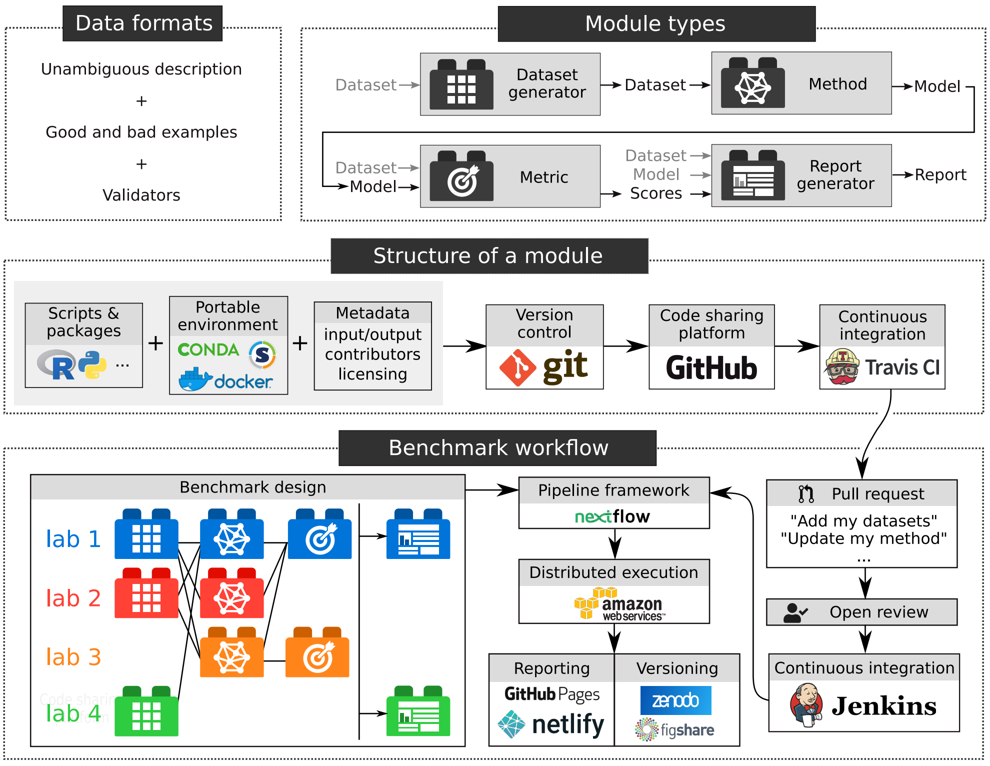


Figure 7.1: The proposed workflow for collaborative and continuous benchmark. **a** A good set of data formats should have an unambiguous description, good and bad examples, and automatic validators. **b** We define several types of modules that load in some data, such as a dataset, and produce a result, such as a model. **c** A module is more than just code: it also defines an environment and metadata, is placed under version control and automatically tested using continuous integration. **d** Modules from different labs or persons are connected through the use of a pipeline framework. A module can be added to the workflow by creating a pull request on the main github repository, followed by review and continuous integration within the framework. The benchmark is executed in parallel and reports are automatically generated from these results.

Data formats

The basis of any collaborative effort in computational biology is agreeing on how data will be interchanged, and benchmarking is no exception. Sometimes, the differences between data formats can be minor, for example whether the samples within a gene expression matrix are put in the rows or in the column. In other cases, different data formats can have a significant impact on storage and/or the speed by which the data can be processed.

In the benchmarking workflow presented here, a format represents how a particular part of a dataset, model, score or report should be represented on disk.

What a format entails

For a format to be useful, it should have an unambiguous description. In this way, someone developing a module can be sure how the inputs look like, even if these inputs do not exist yet, and can also be sure that the outputs will be useful as input for other modules.

While a description is meant to be readable by humans, this description should also be translated into a language computers can understand, so that each data file produced by a module can be validated. In this way, developers of a module can get immediate feedback on whether their output matches the format description. To make a format validatable, it is one possibility to use one of the many "schemas" available, such as json-schema (http://json-schema.org/), XML schemas or Apache Arrow Schemas. Often, there are already validators available for these schemas (json-schemas for example: https://json-schema.org/implementations.html#validators). On the other hand, for more custom file formats or complex behaviour, validators will need to be implemented.

Finally, connecting the human-readable description with the computational validation can be done with providing good and bad examples of the data format. These examples serve a double purpose, because they provide the module contributors several examples to understand the description, but can also be used as test cases for the format validators.

Formats change as the field progresses

Usually, the optimal representation of a dataset or the output of a method only becomes apparent when several methods have been developed already. This means that any effort to make a benchmark collaborative and continuous should strive to make its data formats flexible. Flexibility can take several forms. New features could be added to the format, without invalidating the old data and modules. When this is not an option, new formats could be added alongside the old. When applicable, converters should then be written which convert the old formats into the new, so that old modules keep on functioning. Finally, in extreme cases, old formats could be invalidated and replaced with new formats, which would require some versioning system to make sure modules are run on the version of the formats they were developed.

It is inevitable that disagreements about data representation will emerge in a collaborative effort. But in any case, having a common format, even if they are suboptimal for certain use cases, is usually better than having many disparate formats.

Module types

In the benchmarking workflow, we define four types of modules. In our experience, these four modules are enough to construct a fully comprehensive benchmark of a group of methods.

Dataset generators

This module will generate a dataset, which can from very simple ”toy” data, synthetic data which try to mimic the characteristics of real data as best as possible, or real datasets. Optionally, a dataset generator can use another dataset as input, for example when generating asynthetic dataset based on a real dataset. Only rarely will a dataset generator contain the primary data itself. Rather, data should be gathered directly from primary sources, for example using APIs from the database or by downloading the data directly from data management systems such as Zenodo or Figshare.

Methods

A method module reads in (part) of a dataset, and uses this to generate a model. Some special types of methods can be helpful to include at the start of a benchmark. Positive controls, for example a method that simply return the reference model of the dataset, and negative controls, for example a method that generate a random model, could be useful to make sure the metrics work correctly. Off-the-shelf methods or baseline methods that can be easily implemented with just a few lines of code could be helpful as a reference point to other methods and to assess the difficulty of particular datasets.

A common issue when benchmarking is the selection of algorithm parameters. It is not uncommon that the authors of a method disagree on what parameter settings were used for benchmarking [6]. In our workflow, method authors are required to define for each parameter a default value, but also a distribution of values that can be used for parameter tuning.

Metrics

Metric modules score the output of a model. Some metrics assess the accuracy of a model by comparing it with some reference model present in the dataset. Others will look at the resources consumed by the method, such as CPU time and memory, to assess its scalability. Models can also be compared to other models, for example to examine the stability of a method. Finally, some qualitative metrics can also be defined here, for example those that look at the usability of a method.

Report generators

In the end, the scores are aggregated and interpreted using a report generator. This modules generates a report, which can be static, such as a markdown document with figures, or dynamic in the form of a web application. By crowdsourcing the benchmark interpretation, it would become much less authoritative and instead promotes open discussion in the field [7]. It might certainly happen that different reports would contain contradicting results, but because each reports starts from the same set of data, it would be traceable why the conclusions differ. For example, there might be subtle differences in how the scores have been averaged. Or, a report may only have focused on only a subset of the dataset that the authors found the most relevant for their method.

Modules

Scripts, a portable environment and metadata

A module needs to contain at least one command, which will run some code that reads in the input data, process it in some way, and ultimately write the output data in the correct format.

Given the large diversity of programming languages used in computation biology, a collaborative benchmarking effort should avoid imposing restrictions on the programming language used. As an example, the single-cell analysis field is split between tools written for R and Python [8], and choosing one of these two would therefore alienate a significant part of the field. Moreover, a collaborative effort should be open for new languages, such as Julia [9], which could be more powerful and developer friendly for certain use cases.

Apart from being language agnostic, the execution of a module should also happen on any computer in exactly the same way. To make the execution reproducible, we therefore require that a module defines a portable environment, which contains the necessary operating system, language interpreters and other packages to execute the code within the module. An environment can be portable on many levels: within one programming language such as virtualenv for python or packrat for R or across languages using package managers such as Conda. The most complete level of reproducibility can be obtained by working at the level of the operating system, through container systems such as docker or singularity. Finally, to be able execute stochastic algorithms in a reproducible manner, it is also necessary to fix the pseudo-random number generator in some way, we do this by always setting an a priori defined seed through R or numpy.

A module also contains metadata, which lists the requirements to run the method such as the inputs, outputs and the name of the portable environment. Within our workflow, we also require data for organisational purposes, such as a list of authors with their contributions, and the licence of the code within the module.

Version control and code sharing

We require that the complete module, including the portable environment and metadata, is placed under version control so that any changes are tracked. The module is then shared on a code sharing platform, which makes it possible for other module authors and maintainers of the benchmark to file issues on the module, request some changes to the code through pull requests, and create a modifications if the licence allows it. In our workflow, we use git for version control and GitHub as the platform to share modules, although it should be noted that powerful variants of the latter exist, including self-hosted ones.

Continuous testing and validation

To keep the development of a module and benchmark maintainable, it is important that each element of the module is automatically tested and validated. In this way, many errors are caught early, before they can impact other modules in the benchmark. Including automated testing also reduces the burden for those reviewing the modules. This crosstalk between automated testing and manual reviewing is already commonplace in many package repositories, such as CRAN and Bioconductor.

In our proposed workflow, we automatically trigger a new test on (https://www.travis-ci.com), which is cost-free for open-source projects. We test each module on several levels. We first check whether it contains all required content, and whether the metadata is complete. Next, we activate the portable environment, run the module on some small input data, and validate the produced output. If any of these steps fail, the author is notified. Only when tests are successful can the new module be integrated into the whole benchmark procedure.

Combining modules within a benchmark

To make the benchmark as inclusive as possible, it should be possible for anyone to extend or adapt the benchmark for their own purposes. At the same time, it would also be useful to have a central place that lists all the modules and provides the most up-to-date set of reports for interested readers. To reconcile these two criteria, our benchmarking workflow has one ”main” repository, which lists the location of the different modules and how they are combined in the benchmark. Anyone can create a fork of this repository, adapt the modules or benchmarking design in any way, and run it using their own infrastructure.

Executing the benchmark

For the execution of the modules, a pipeline manager such as snakemake [10] or nextflow [11] is almost indispensable. These tools make sure the modules are executed in the correct order and within a reproducible environment. Moreover, to make the benchmark scalable, a pipeline manager will only rerun those executions for which inputs have changed, including changes to scripts or packages inside the portable environment. Within our benchmarking workflow, we created a custom pipeline manager for this, which provided us with features that are lacking in most current pipeline managers, such as incrementality at the level of the portable environment, output validation and fixation of the pseudo-random number generator.

Adding or updating a module

While anyone is able to fork and modify the benchmark repository, modifications to the main repository, such as additions or updates of a module, still requires some form of control by a group of maintainers. This group of maintainers, which would primarily consist of authors of other modules, are responsible for checking whether the module has passed all automated checks. and give feedback regarding data formats and testing results. Notably, reviewing happens in a completely open fashion, similar as to what is done in open-source communities, such as Bioconductor and ropensci (https://github.com/ropensci/onboarding).

We expect many different researchers to contribute to the benchmark. Some may just want to contribute some data, and can thus create a very light-weight dataset generator module. Developers may just want to integrate their own method, but may make use of the existing datasets, metrics and reports. Researchers who are interested in benchmarking may contribute some (synthetic) dataset generators, metrics and a report summarizing their main findings. Such reports may also be generated by a group of people who met at a workshop or conference.

In our workflow, adding or updating a module can be done by cloning the repository, making the necessary changes, and then creating a pull request on Github.

Continuous benchmarking and versioning

Every time the main repository is updated, for example with a new version of a module, an update of the whole benchmark workflow is triggered. Only those modules with outdated input, because the code, environment or some input files changed, are rerun.

The end results of the benchmark are one or more reports, which are freely accessible online. At regular time intervals, such as monthly, all the reports can be gathered and released as a new ”version”, which includes a changelog of updates to any of the modules made before the last release. This release is given a digital object identifier and registered at an open-access repository such as zenodo or figshare.

Possible issues

A continuous and collaborative benchmarking effort may sound great in theory, but it will undoubtedly come across many issues. These are mainly centred around having correct incentives, because if nobody has to gain something by joining the benchmark, it will die just like many efforts before it.

There are many reasons why someone that creates a method would have no (or even negative) incentives to participate in a community-wide benchmark. It might be too much hassle to comply with the standards of the project, including data formats and continuous integration. Containers already solve part of this problem, because they allow the developer to have a ”sandbox” in their own module. Furthermore, clear documentation and sufficient automation should make it very easy for a developer to include a new module.

Another negative incentive may be that the developer is scared that their method won't come out on top, and will therefore be obsolete even before it is published. The problem here is not the developer, but rather the field-wide focus (or sometimes obsession) with being on top of the ranking in a benchmarking study. Creating objective metrics and aggregating the scores is very difficult, and small changes can easily (slightly) change the ranking. What is most important is not that the method is the new overall best (even if the performance increase is negligible), but that it outperforms other methods for some use cases or metrics, such as finding the right topology in the case of trajectory inference. Allowing the developer to create their own metrics and reports might partially mitigate this issue. But the best way to resolve this might be to change the mindset in the bio-informatics community away from obsessions with rankings, and towards a more gentle context-dependent view on method performance.

In the end, the best way to overcome these negative incentives is to create a large positive incentive: saving the precious time of developers and giving them a venue to publicize their method to the bio-informatics community. To get to this point, the benchmark would need to reach a critical mass, containing some datasets, metrics and the current state-of-the-art methods.

Another issue with a continuous benchmarking effort will be the long-term sustainability. Often, there is no incentive to further support a bio-informatics tool after it was published [1]. But as the field changes, so should the benchmark, and thus its metrics, datasets, data formats and reports. After a while, some "code rot" [2] may be accumulating, with report generators or methods no longer working correctly. Resolving this issue will be difficult, and it is hard to estimate its impact beforehand.

Conclusion and outlook

Continuous and collaborative benchmarking provides an alternative and powerful way to evaluate methods in computational biology. It relies heavily on modularisation and tools from software development, which make it possible to design a benchmarking strategy that can be easily extended by anyone, while still allowing for open discussion to exist in a field. Because less effort is spent developing new benchmarking pipelines for every new method, this workflow would speed up method development in bioinformatics. Because the result of the benchmark can be easily interpreted by anyone, it would also avoid other issues, such as the self-assessment trap [12].

[1]Perhaps one of the reasons why only 30% of the authors responded to our feedback within the trajectory inference benchmark

[2]A term from software development describing the tendency of software to slowly stop working if it is not updated due to changes in dependencies and the operating system

As in every collaborative effort, a benchmarking workflow like this will have to find a balance between quantity and quality. On the one hand do we want to include anyone willing to make an effort to add something to the benchmark. At the same time do we not want to be overloaded with support requests. One way to find this balance is to look at similar projects outside of benchmarking, at vibrant package repositories (CRAN, pypi, bioconductor, ropensci, npm). These projects make the entrance barrier low through in-depth documentation and "skeletons", by which a developer can quickly get going and test things out. skeletons and documentation. Their long-term success depends on the strict enforcements community standards (such as automated testing), which is semi-automated to keep the need for human reviewers low.

In the ideal case, a continuous benchmarking project should be supported by a larger consortium, such as the Human Cell Atlas, which would not only assure its continuity, but would also provide infrastructural support. In particular, services that have strong requirements on the side of storage and/or computing power would benefit from this, such as continuous integration, the code sharing platform and execution environment. If supported by these organisations, continuous and collaborative benchmarking might have a tremendous impact on the speed by which new methods are developed in bioinformatics, and as a result on our understanding of biology.

References

[1] Junyue Cao et al. “The Single-Cell Transcriptional Landscape of Mammalian Organogenesis”. In: *Nature* 566.7745 (Feb. 2019), p. 496. ISSN: 1476-4687. DOI: 10.1038/s41586-019-0969-x.

[2] Patrick K. Kimes and Alejandro Reyes. “Reproducible and Replicable Comparisons Using SummarizedBenchmark”. In: *Bioinformatics* 35.1 (Jan. 1, 2019), pp. 137–139. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bty627.

[3] Charlotte Soneson and Mark D. Robinson. “iCOBRA: Open, Reproducible, Standardized and Live Method Benchmarking”. In: *Nature Methods* 13.4 (Apr. 2016), p. 283. ISSN: 1548-7105. DOI: 10.1038/nmeth.3805.

[4] John Moult et al. “Critical Assessment of Methods of Protein Structure Prediction (CASP)—Round XII”. In: *Proteins: Structure, Function, and Bioinformatics* 86.S1 (2018), pp. 7–15. ISSN: 1097-0134. DOI: 10.1002/prot.25415.

[5] Mingxiang Teng et al. “A Benchmark for RNA-Seq Quantification Pipelines”. In: *Genome Biology* 17.1 (Apr. 23, 2016), p. 74. ISSN: 1474-760X. DOI: 10.1186/s13059-016-0940-1.

[6] Qiwen Hu and Casey S. Greene. “Parameter Tuning Is a Key Part of Dimensionality Reduction via Deep Variational Autoencoders for Single Cell RNA Transcriptomics”. In: *bioRxiv* (Sept. 20, 2018), p. 385534. DOI: 10.1101/385534.

[7] Raphael Silberzahn and Eric L. Uhlmann. “Crowdsourced Research: Many Hands Make Tight Work”. In: *Nature* 526.7572 (Oct. 8, 2015), pp. 189–191. ISSN: 1476-4687. DOI: 10.1038/526189a.

[8] Luke Zappia, Belinda Phipson, and Alicia Oshlack. “Exploring the Single-Cell RNA-Seq Analysis Landscape with the scRNA-Tools Database”. In: *bioRxiv* (Oct. 20, 2017), p. 206573. DOI: 10.1101/206573.

[9] Jeff Bezanson et al. “Julia: A Fresh Approach to Numerical Computing”. In: (Nov. 6, 2014).

[10] Johannes Köster and Sven Rahmann. “Snakemake—a Scalable Bioinformatics Workflow Engine”. In: *Bioinformatics* 28.19 (Oct. 1, 2012), pp. 2520–2522. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/bts480.

[11] Paolo Di Tommaso et al. “Nextflow Enables Reproducible Computational Workflows”. In: *Nature Biotechnology* 35.4 (Apr. 2017), p. 316. ISSN: 1546-1696. DOI: 10.1038/nbt.3820.

[12] Raquel Norel, John Jeremy Rice, and Gustavo Stolovitzky. “The Self-Assessment Trap: Can We All Be Better than Average?” In: *Molecular Systems Biology* 7 (Oct. 11, 2011), p. 537. ISSN: 1744-4292. DOI: 10.1038/msb.2011.70.

Appendices

A | CV

This appendix contains my scientific curriculum vitae.

Wouter Saelens

PhD student
Inflammation Research Center, a VIB-UGent Department, Technologiepark-Zwijnaarde 71, 9052 Zwijnaarde, Belgium
+32 485 81 40 59 | *wouter.saelens@gmail.com* | zouter | zouters

Education

Ghent University

M.Sc. Biochemistry & Biotechnology *2012-2014*

- Major bioinformatics and systems biology
- Minor plant biotechnology
- Thesis: Lokale celtype-specifieke genexpressie in het myeloïde transcriptoom
- Thesis advisors: Bart N. Lambrecht and Yvan Saeys

VIB - Ghent University

Doctor of Science: Bioinformatics *2014-2019*

- Developing and benchmarking methods for analysing transcriptomics data
- Promoters: Yvan Saeys (VIB - UGent) and Bart N. Lambrecht (VIB - UGent)
- Defending October 24

Grants

2014-2015	BOF PhD grant
2015-2019	FWO Aspirant (extended in 2017)
2018	FWO Travel grant: 3 month visit to the Mark Robinson lab, University of Zürich

Research

Primary research

Robin, B., **Saelens, W.***, Saeys, Y.*, Accepted. NicheNet: Modeling intercellular communication by linking ligands to target genes. Nature Methods. ** Co-supervised the work*

Weber, L.M., **Saelens, W.**, Cannoodt, R., Soneson, C., Hapfelmeier, A., Gardner, P.P., Boulesteix, A.-L., Saeys, Y., Robinson, M.D., 2019. Essential guidelines for computational method benchmarking. Genome Biology 20. *https://doi.org/10.1186/s13059-019-1738-8*

Saelens, W.*, Cannoodt, R.*, Todorov, H., Saeys, Y., 2019. A comparison of single-cell trajectory inference methods. Nature Biotechnology 37, 547–554. ** Shared first author https://doi.org/10.1038/s41587-019-0071-9*

Saelens, W., Cannoodt, R., Saeys, Y., 2018. A comprehensive evaluation of module detection methods for gene expression data. Nature Communications 9. *https://doi.org/10.1038/s41467-018-03424-4*

Scott, C.L., T'Jonck, W., Martens, L., Todorov, H., Sichien, D., Soen, B., Bonnardel, J., De Prijck, S., Vandamme, N., Cannoodt, R., **Saelens, W.**, Vanneste, B., Toussaint, W., De Bleser, P., Takahashi, N., Vandenabeele, P., Henri, S., Pridans, C., Hume, D.A., Lambrecht, B.N., De Baetselier, P., Milling, S.W.F., Van Ginderachter, J.A., Malissen, B., Berx, G., Beschin, A., Saeys, Y., Guilliams, M., 2018. The Transcription Factor ZEB2 Is Required to Maintain the Tissue-Specific Identities of Macrophages. Immunity 49, 312–325.e5. *https://doi.org/10.1016/j.immuni.2018.07.004*

Scott, C.L., Soen, B., Martens, L., Skrypek, N., **Saelens, W.**, Taminau, J., Blancke, G., Van Isterdael, G., Huylebroeck, D., Haigh, J., Saeys, Y., Guilliams, M., Lambrecht, B.N., Berx, G., 2016. The transcription factor Zeb2 regulates development of conventional and plasmacytoid DCs by repressing Id2. The Journal of Experimental Medicine 213, 897–911. *https://doi.org/10.1084/jem.20151715*

Van de Laar, L., **Saelens, W.**, De Prijck, S., Martens, L., Scott, C.L., Van Isterdael, G., Hoffmann, E., Beyaert, R., Saeys, Y., Lambrecht, B.N., Guilliams, M., 2016. Yolk Sac Macrophages, Fetal Liver, and Adult Monocytes Can Colonize an Empty Niche and Develop into Functional Tissue-Resident Macrophages. Immunity 44, 755–768. *https://doi.org/10.1016/j.immuni.2016.02.017*

Preprints

Van den Berge, K., de Bézieux, H.R., Street, K., **Saelens, W.**, Cannoodt, R., Saeys, Y., Dudoit, S., Clement, L., 2019. Trajectory-based differential expression analysis for single-cell sequencing data. *https://doi.org/10.1101/623397*

Saelens, W.*, Cannoodt, R.*, Todorov, H., Saeys, Y., 2018. A comparison of single-cell trajectory inference methods: towards more accurate and robust tools. ** Shared first author https://doi.org/10.1101/276907*

Cannoodt, R., **Saelens, W.**, Sichien, D., Tavernier, S., Janssens, S., Guilliams, M., Lambrecht, B.N., De Preter, K., Saeys, Y., 2016. SCORPIUS improves trajectory inference and identifies novel modules in dendritic cell development. *https://doi.org/10.1101/079509*

Review articles

Cannoodt, R.*, **Saelens, W.***, Saeys, Y., 2016. Computational methods for trajectory inference from single-cell transcriptomics. European Journal of Immunology 46, 2496–2506. ** Shared first author https://doi.org/10.1002/eji.201646347*

Book chapters

Todorov, H., Cannoodt, R., **Saelens, W.**, Saeys, Y., 2018. Network Inference from Single-Cell Transcriptomic Data. Gene Regulatory Networks 235–249. *https://doi.org/10.1007/978-1-4939-8882-2_10*

Teaching

Teaching assistant

Course 'Programming' *2016, 2017, 2018*
- Helping Biology and Biochemistry & Biotechnology students to learn basic programming

Bachelor's project *2015, 2016, 2017, 2018*
- Guiding groups of 3 Biochemistry & Biotechnology students do a research project within one week

Master thesis guidance

2014-2015	Alexander Reinartz: Zoektocht naar de drijvende krachten achter macrofaag- en Kupffercel-identiteit
2016-2017	Robin Browaeys: Linking extracellular signals to target genes by data integration
2017-2018	Chloë Guidi: Improving regulatory network inference based on single-cell pseudotime data

Master projects and internship guidance

2015	Charlotte De Vogelaere
2015	Solène Boutin
2016	Annelies Emmaneel

Conferences and workshops

Invited talks

2019	Single-cell analysis SIB - SciLifeLab Autumn School	*Leysin, CH*
2019	SincellTE, 5 day single-cell analysis course	*Station Biologique Roscoff, FR*
2019	EPFL Single-cell mini symposium	*EPFL Lausanne, CH*

Accepted talks

2019	Keystone Single-cell biology: Infering trajectories using dyno	*Breckenridge, CO, US*
2015	Benelux bioinformatics conference: A comprehensive comparison of module detection methods	*Antwerp, BE*

Accepted posters

2019	Keystone Single-cell biology: Infering trajectories using dyno	*Breckenridge, CO, US*
2018	Single-cell genomics: Evaluating single-cell trajectory inference methods using simulated data	*Hinxton, UK*
2017	Keystone Single-cell -omics: Simulation of realistic single-cell data to promote the development of new modelling methods	*Stockholm, SE*
2016	Single-cell genomics: Inferring trajectories using SCORPIUS	*Hinxton, UK*
2016	Benelearn: A comprehensive evaluation of module detection methods for gene expression data	*Kortrijk, BE*
2016	Keystone Systems immunology: Inferring branched trajectories from single cell data reveals novel insights into immune cell differentiation	*Big Sky, MT, US*
2015	Benelux Bioinformatics conference: A comprehensive evaluation of module detection methods for gene expression data	*Antwerp, BE*
2014	Benelux BIoinformatics conference: Biclustering by taking into account the relationships between conditions and application on immunological expression data	*Luxembourg, LU*

Awards

2015	Best oral presentation at Benelux Bioinformatics conference	*Antwerp, BE*
2014	Bayer prize meritorious student (Bayer prijs voor verdienstelijke student)	

Service

Public outreach

Wetenschap op stap *2016, 2017, 2018, 2019*
- One day of teaching and doing experiments with school children in 6th grade about biology and bioinformatics

WeGoStem *2018*
- One day of teaching and doing experiments with school children in 5-6th grade about robotics

Reviewed for

Bioinformatics
Frontiers in Immunology
Nucleic Acids Research

B | Vignette of *dyno*, A toolkit for inferring trajectories

This appendix contains the user guide for the dyno package, which groups several other packages useful for inferring and interpreting trajectories. This documentation is also available at https://dynverse.org/users/3-user-guide/.

```
library(dyno)
library(tidyverse)
```

Preparing the data

Gene expression data

As input, dynwrap requires raw counts and normalised (log2) expression data. Cells with low expression, doublets and other "bad" cells should already be filtered from this matrix. Features (i.e. genes) may already be filtered, but this is not required. Some methods internally include a feature filtering step, while others can handle a lot of features just fine.

Internally, dynwrap works with a sparse matrix (`dgCMatrix`) which reduces the memory footprint.

```
dataset <- wrap_expression(
  expression = example_dataset$expression,
  counts = example_dataset$counts
)
```

Prior information

Some methods require prior information to be specified. You can add this prior information to the dataset using `dynwrap::add_prior_information`:

```
dataset <- add_prior_information(
  dataset,
  start_id = "Cell1"
)
```

Optional information

Grouping / clustering

You can add a grouping or clustering to the data using `dynwrap::add_grouping`:

```
dataset <- add_grouping(
  dataset,
  example_dataset$grouping
)
```

Dimensionality reduction

You can add a grouping or clustering to the data using `dynwrap::add_dimred`. The dimensionality reduction should be a matrix with the same rownames as the original expression matrix.

```
dataset <- add_dimred(
  dataset,
  example_dataset$dimred
)
```

Current limitations

Currently, alternative input data such as ATAC-Seq or cytometry data are not yet supported, although it is possible to simply include this data as expression and counts.

In the near future, we will also add the ability to include RNA velocity as input. See the discussion at https://github.com/dynverse/dynwrap/issues/112

Selecting the best methods for a dataset

Within our evaluation study, we compared 45 methods on four aspects:

- **Accuracy**: How similar is the inferred trajectory to the "true" (or "expected") trajectory in the data. We used several metrics for this, comparing the cellular ordering and topology, and compared against both real datasets, for which a gold standard is not always so well defined, and synthetic data, which are not necessarily as biologically relevant as real data.
- **Scalability**: How long the method takes to run and how much memory it consumes. This mainly depends on the dimensions of the input data, i.e. the number of cells and features.
- **Stability**: How stable the results are when rerunning the method with different seeds or slightly different input data.
- **Usability**: The quality of the documentation and tutorials, how easy it is to run the method, whether the method is well tested, … We created a transparent scoresheet to assess each of these aspects in a *more or less* objective way.

Perhaps not surprisingly, we found a high diversity in method performance, and that not many methods perform well across the board. The performance of a method depended on many factors, mainly the dimensions of the data and the kind of trajectory present in the data. Based on this, we developed an interactive shiny app which you can use to explore the results and select an optimal set of methods for your analysis.

This app can be opened using `dynguidelines::guidelines_shiny()`. It is recommended to give this function your dataset, so that it will precalculate some fields for you:

```r
dataset <- example_dataset
guidelines_shiny(dataset = dataset)
```

The app includes a tutorial, which will guide you through the user interface. Once finished, it is highly recommended to copy over the code that generates the guidelines to your script, so that your analysis remains reproducible, for example:

```r
dataset <- example_dataset
guidelines <- guidelines(
  dataset,
  answers = answer_questions(
    dataset,
    multiple_disconnected = FALSE,
    expect_topology = TRUE,
    expected_topology = "linear"
  )
)
```

```
## Loading required namespace: akima
```

This guidelines object contains:

- Information on the selected methods: `guidelines$methods`
- The names of the selected methods: `guidelines$methods_selected`
- The answers given in the app (or their defaults): `guidelines$answers`

Inferring trajectories

`dynwrap::infer_trajectory` is the main function to infer a trajectory. It requires two things:

- A dataset, wrapped using `dynwrap::wrap_expression`
- A TI method. This can be one of the 59 TI method from dynmethods, or a name of a method in which case it will retrieve the relevant method from dynmethods.

```
dataset <- wrap_expression(
  counts = example_dataset$counts,
  expression = example_dataset$expression
)
model <- infer_trajectory(dataset, ti_comp1())
```

```
## Loading required namespace: hdf5r
```

This model now contains the main information on the trajectory, i.e. the `milestone_network` and `progressions`:

```
model$milestone_network
```

```
##              from           to length directed
## 1 milestone_begin milestone_end      1    FALSE
```

```
head(model$progressions, 10)
```

```
##    cell_id            from            to percentage
## 1    Cell1 milestone_begin milestone_end 0.25088041
## 2    Cell2 milestone_begin milestone_end 0.36976202
## 3    Cell3 milestone_begin milestone_end 0.56873343
## 4    Cell4 milestone_begin milestone_end 0.91506325
## 5    Cell5 milestone_begin milestone_end 0.21259337
## 6    Cell6 milestone_begin milestone_end 0.91000325
## 7    Cell7 milestone_begin milestone_end 0.90920327
## 8    Cell8 milestone_begin milestone_end 0.64542720
## 9    Cell9 milestone_begin milestone_end 0.60957610
## 10  Cell10 milestone_begin milestone_end 0.06369673
```

While running methods inside a docker or singularity container reduces problems with dependencies and makes an analysis more reproducible, it can also create a considerable overhead. We plan to wrap wrap some more "popular" methods directly into R. See https://github.com/dynverse/dynmethods/issues/152 for an overview.

Parameters

Optionally, you can also give it some parameters. These are all documented within the relevant functions in dynmethods (also available in the reference section):

```
?ti_comp1
```

```
## Component 1
##
## Description:
##
##      Will generate a trajectory using Component 1.
##
##      This method was wrapped inside a container.
```

```
##
## Usage:
##
##      ti_comp1(dimred = "pca", ndim = 2L, component = 1L)
##
## Arguments:
##
##   dimred: Which dimensionality reduction method to use. Domain: pca,
##           mds, tsne, ica, lle, landmark_mds, mds_sammon, mds_isomds,
##           mds_smacof, umap, dm_diffusionMap. Default: pca. Format:
##           character.
##
##     ndim: . Domain: U(2, 30). Default: 2. Format: integer.
##
## component: . Domain: U(1, 10). Default: 1. Format: integer.
##
## Value:
##
##      A TI method wrapper to be used together with 'infer_trajectory'
```

Visualisng a trajectory

The main functions for plotting a trajectory are included in the dynplot package.

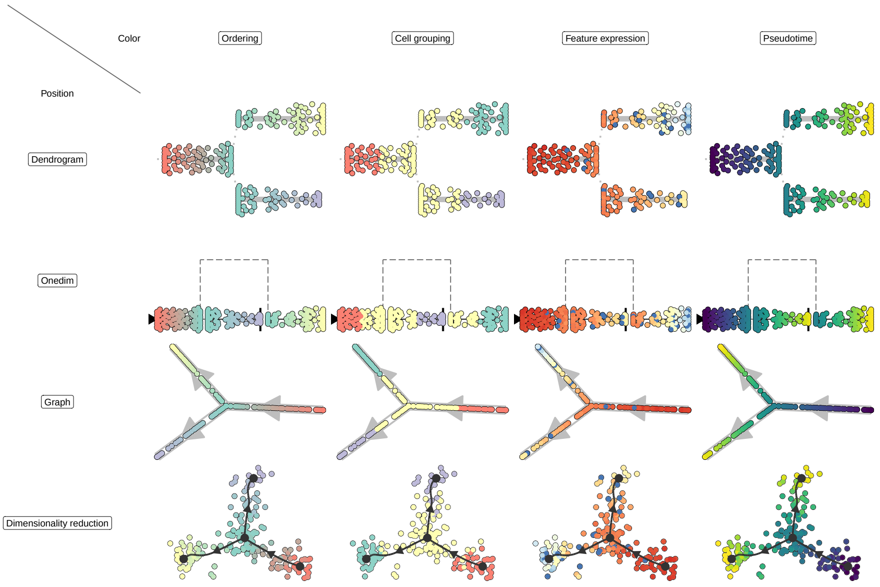
We'll use an example toy dataset

```r
set.seed(1)
dataset <- dyntoy::generate_dataset(model = "bifurcating", num_cells = 200)
```

To visualise a trajectory, you have to take into acount two things:

- Where will I place the trajectory and cells in my 2D space
- What do I want to visualise along the trajectory based on color

Depending on the answer on these two questions, you will need different visualisations:

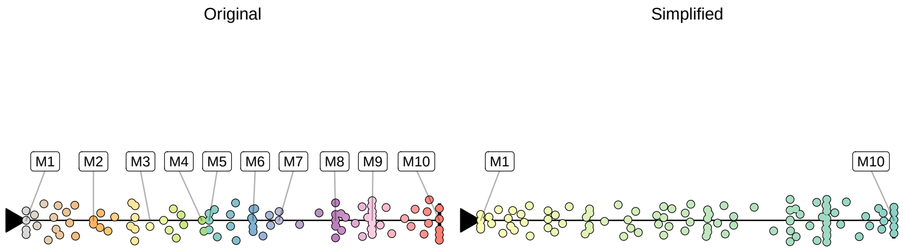


Adapting the trajectory

Simplifying

Intermediate milestones can be removed by simplyfing the trajectory:

```r
model <- dyntoy::generate_dataset(model = dyntoy::model_linear(num_milestones = 10))
simplified <- simplify_trajectory(model)
```



Rooting

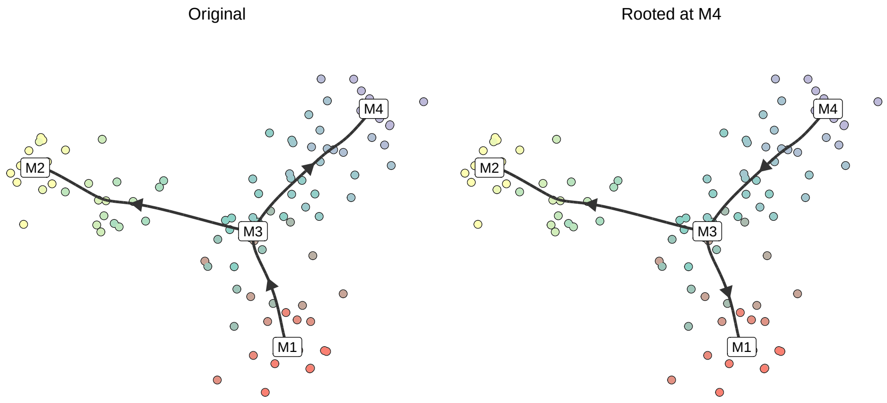
TI methods often do not have an idea where the root of a trajectory is. We provide two ways of rooting a trajectory. After rooting, all other edges will point away from the root.

```
set.seed(1)
model <- dyntoy::generate_dataset(model = dyntoy::model_bifurcating())
```

Manually

If you know the milestone (or cell) that is at the start of the trajectory, you can directly call `add_root`:

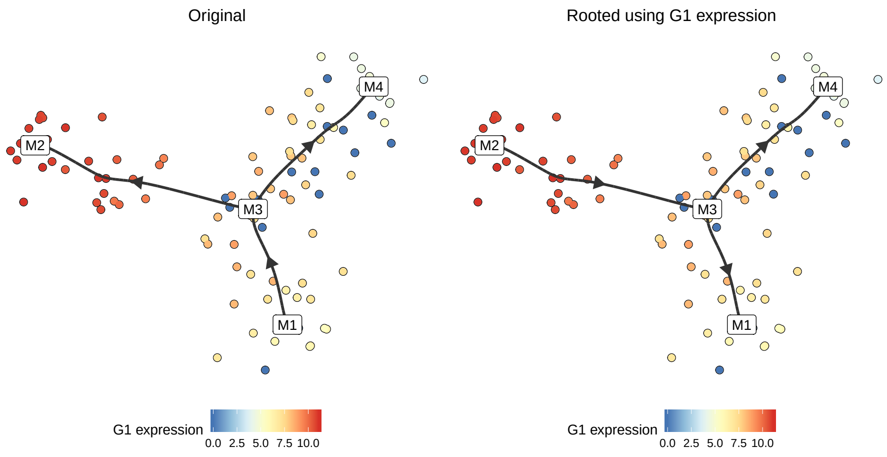
```
model_rooted <- model %>% add_root(root_milestone_id = "M4")
```



Using marker genes

If you know some marker genes that are highly expressed at the start of the trajectory, rooting can be done implicitely:

```
model_rooted <- model %>% add_root_using_expression("G1", expression_source = model)
```



Rooting a trajectory based on RNA velocity is on our todo list. See https://github.com/dynverse/dynwrap/issues/115

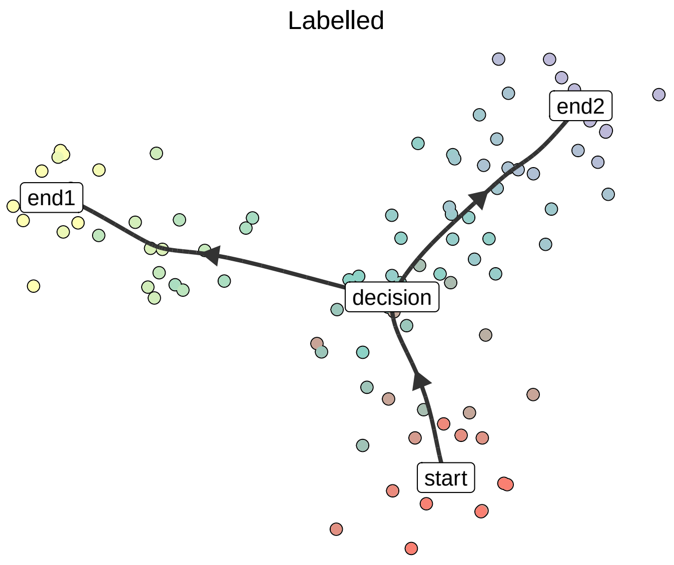
Annotating

Annotating/labelling milestones is still experimental

Similarly as with rooting, there are also two ways to annotate the milestones within a trajectory:

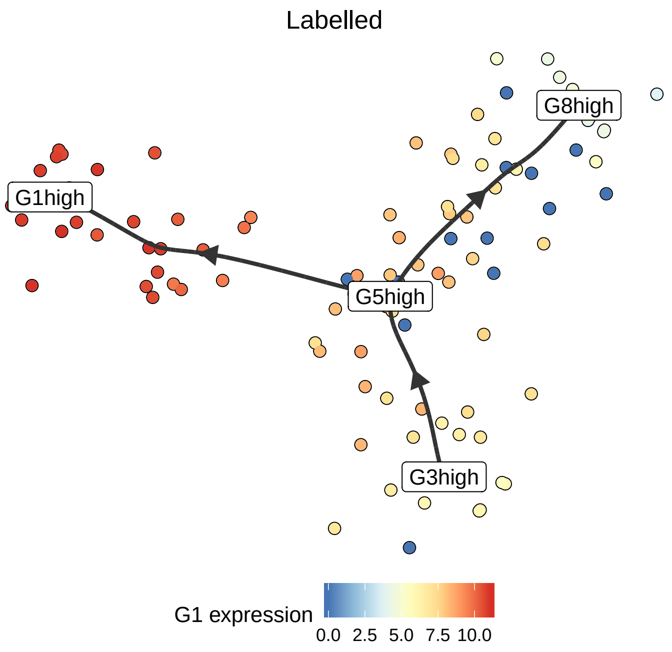
Manually

```
model_labelled <- model %>%
  label_milestones(c(M1 = "start", M2 = "end1", M3 = "decision", M4 = "end2"))
```



Using marker genes

```r
model_labelled <- label_milestones_markers(
  model,
  markers = list(
    G1high = c("G1"),
    G5high = c("G5"),
    G3high = c("G3"),
    G8high = c("G8")
  )
)
```



Annotating milestones based on external information is on our todo list.

Limitations

Splitting a trajectory and adding intermediate milestones is on our todo list.

Trajectory differential expression

Compared to differential expression between clusters of cells, defining differential expression on trajectories is not so straightforward. What constitutes a trajectory differentially expressed gene?

- A gene that is uniquely expressed in a particular branch?
- A gene that changes at a branching point?
- A gene that changes along pseudotime?
- …?

dynfeature is a package that allows you to find these different kinds of differential expression in a trajectory. It first defines a particular variable that needs to be predicted (for example, whether a cell is present in a branch or not), and tries to predict this variable based on the expression in different cells. It then ranks each feature based on their predictive capability, and based on this ranking you can select differentially expressed genes.

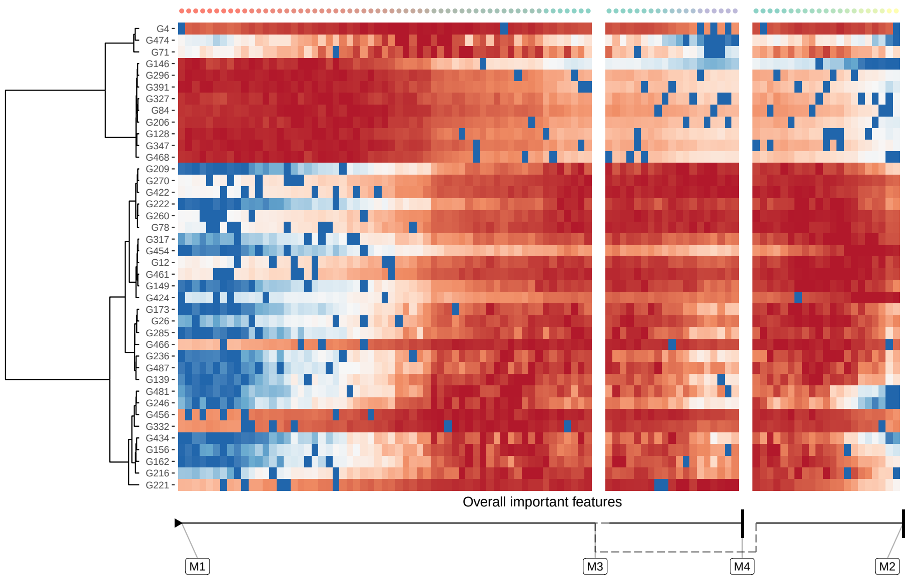
Depending on what variable is predicted, you get a different ranking. This simply depends on what kind of features you are interested in:

A global overview of the most predictive genes

If you just want to see features that change anywhere in the trajectory, you can use 'dynfeature::

```
model <- dyntoy::generate_dataset(model = dyntoy::model_bifurcating(), num_features = 500)
```

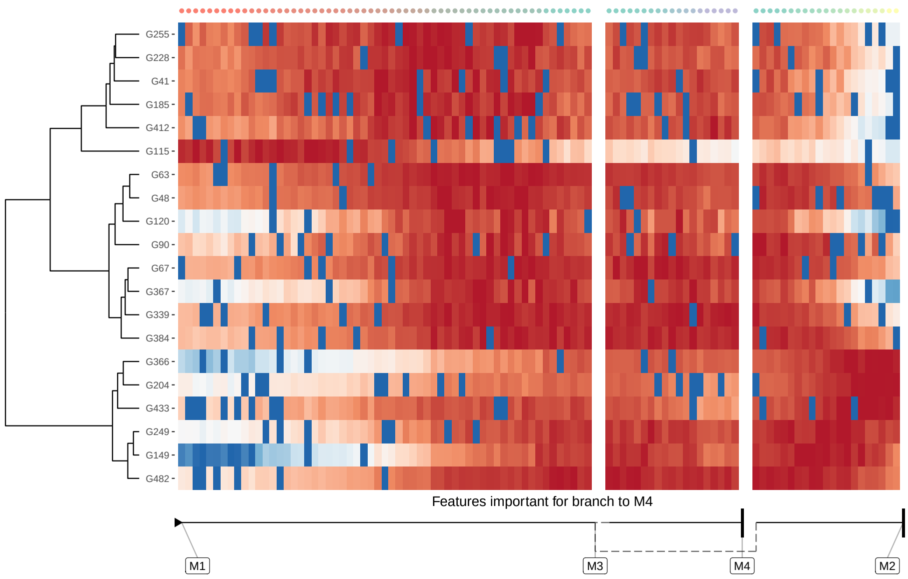
```
overall_feature_importances <- dynfeature::calculate_overall_feature_importance(model)
features <- overall_feature_importances %>%
  top_n(40, importance) %>%
  pull(feature_id)
```



Lineage/branch markers

We can also extract features that are specifically upregulated or downregulated in a specific branch:

```
branch_feature_importance <- calculate_branch_feature_importance(model)
features <- branch_feature_importance %>%
  filter(to == "M4") %>%
  top_n(20, importance) %>%
  pull(feature_id)
```

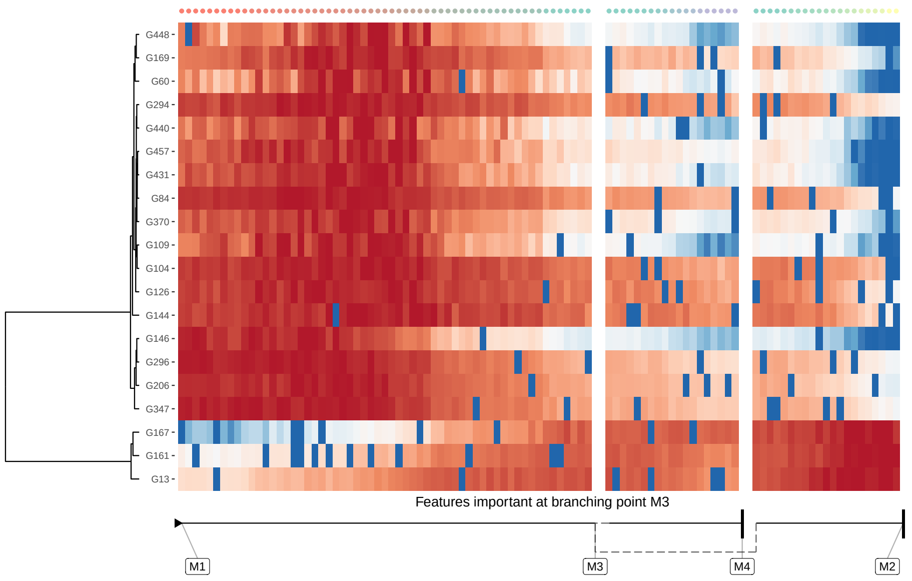


Genes important at bifurcation points

We can also extract features which change at the branching point

```
branching_milestone <- "M3"
branch_feature_importance <- calculate_branching_point_feature_importance(
  model,
  milestones_oi = branching_milestone
  )

features <- branch_feature_importance %>% top_n(20, importance) %>% pull(feature_id)
```



Current limitations

While dynfeature is useful to rank the features according to the strength of trajectory differential expression, they do not provide a statistical ground to find features which are significantly differentially expressed.