Understanding cellular differentiation by modelling of single-cell gene expression data

Dissertation

for the award of the degree
"Doctor of Philosophy"
Division of Mathematics and Natural Sciences
of the Georg-August-Universität Göttingen

within the doctoral program IMPRS PBCS
of the Georg-August University School of Science (GAUSS)

submitted by
Nikolaos Papadopoulos
from Thessaloniki, Greece
Göttingen, 2019

Thesis Committee

- Dr. Johannes Soeding, Research Group Quantitative and Computational Biology, Max Planck Institute for Biophysical Chemistry
- Dr. Melina Schuh, Department Meiosis, Max Planck Institute for Biophysical Chemistry
- Dr. Michael Habeck, Research Group Statistical Inverse Problems in Biophysics, Max Planck Institute for Biophysical Chemistry

Members of the Examination Board

Referee: Dr. Johannes Soeding, Research Group Quantitative and Computational Biology, Max Planck Institute for Biophysical Chemistry

2nd Referee: Dr. Melina Schuh, Department Meiosis, Max Planck Institute for Biophysical Chemistry

Further members of the Examination Board

- Dr. Michael Habeck, Research Group Statistical Inverse Problems in Biophysics, Max Planck Institute for Biophysical Chemistry
- Dr. Juliane Liepe, Research Group Quantitative and Systems Biology, Max Planck Institute for Biophysical Chemistry
- Prof. Dr. Burkhardt Morgenstern, Institute for Microbiology and Genetics, Department Bioinformatics, Georg-August University Göttingen
- Dr. Nico Posnien, Johann Friedrich Blumenbach Institute of Zoology and Anthropology, Department of Developmental Biology, Göttingen Center for Molecular Biosciences, Georg-August University Göttingen

Date of oral examination: 08.08.2019

Summary

Over the course of the last decade single-cell RNA sequencing (scRNA-seq) has revolutionized the study of cellular heterogeneity, as one experiment routinely covers the expression of thousands of genes in tens or hundreds of thousands of cells. By quantifying differences between the single cell transcriptomes it is possible to reconstruct the process that gives rise to different cell fates from a progenitor population and gain access to trajectories of gene expression over developmental time. Tree reconstruction algorithms must deal with the high levels of noise, the high dimensionality of gene expression space, and strong non-linear dependencies between genes.

In this thesis we address three aspects of working with scRNA-seq data: (1) lineage tree reconstruction, where we propose MERLoT, a novel trajectory inference method, (2) method comparison, where we propose PROSSTT, a novel algorithm that simulates scRNA-seq count data of complex differentiation trajectories, and (3) noise modelling, where we propose a novel probabilistic description of count data, a statistically motivated local averaging strategy, and an adaptation of the cross validation approach for the evaluation of gene expression imputation strategies. While statistical modelling of the data was our primary motivation, due to time constraints we did not manage to fully realize our plans for it.

Increasingly complex processes like whole-organism development are being studied by single-cell transcriptomics, producing large amounts of data. Methods for trajectory inference must therefore efficiently reconstruct *a priori* unknown lineage trees with many cell fates. We propose MERLoT, a method that can reconstruct trees in sub-quadratic time by utilizing a local averaging strategy, scaling very well on large datasets. MERLoT compares favorably to the state of the art, both on real data and a large synthetic benchmark.

The absence of data with known complex underlying topologies makes it challenging to quantitatively compare tree reconstruction methods to each other. PROSSTT is a novel algorithm that simulates count data from complex differentiation processes, facilitating comparisons between algorithms. We created the largest synthetic dataset to-date, and the first to contain simulations with up to 12 cell fates. Additionally, PROSSTT can learn simulation parameters from reconstructed lineage trees and produce cells with expression profiles similar to the real data.

Quantifying similarity between single-cell transcriptomes is crucial for clustering scRNA-seq profiles to cell types or inferring developmental trajectories, and appropriate statistical modelling of the data should improve such similarity calculations. We propose a Gaussian mixture of negative binomial distributions where gene expression variance depends on the square of the average expression. The model hyperparameters can be learned via the hybrid Monte Carlo algorithm, and a good initialization of average expression and variance parameters can be obtained by trajectory inference.

A way to limit noise in the data is to apply local averaging, using the nearest neighbours of each cell to recover expression of non-captured mRNA. Our proposal, nearest neighbour smoothing with optimal

bias-variance trade-off, optimizes the k-nearest neighbours approach by reducing the contribution of inappropriate neighbours. We also propose a way to assess the quality of gene expression imputation. After reconstructing a trajectory with imputed data, each cell can be projected to the trajectory using non-overlapping subsets of genes. The robustness of these assignments over multiple partitions of the genes is a novel estimator of imputation performance.

Finally, I was involved in the planning and initial stages of a mouse ovary cell atlas as a collaboration.

Acknowledgements

During my PhD I grew tremendously as a scientist and as a person, if I do say so myself, and none of this would have been possible without all the people that I met on the way, before or after starting. This section should be about a page long, so, sadly, I can't name everyone. Still, thank you all.

Johannes, for giving me this chance, showing me his curiosity, teaching me how he thinks, pushing me when I wasn't trying hard enough, explaining that sometimes it's all smoke and mirrors, and being genuinely nice.

Dr. Melina Schuh and Dr. Michael Habeck, for the co-supervision, the nice discussions, pertinent questions, and good ideas.

Dr. Juliane Liepe, Prof. Dr. Burkhard Morgenstern, and Dr. Nico Posnien, for agreeing to be part of the examination committee.

Janine and Kirsten, for making sure things ran smoothly.

Christian, for annoying me with superior programming, statistics, and data science knowledge until I was passably good, and also for all the discussions, work-related or not, the self-deprecating humour, and John Oliver.

Milot, for his technical expertise, willingness to help, and being Friend.

Gonzalo, for showing me the academia ropes and challenging me to improve, work hard, and set boundaries.

Wanwan, Salma, Annika, Ruoshi (sorry about the interview!), Franco, Saikat, and Eli, as well as our Master students and Hiwis (shout-out to Matt for giving me a living, breathing person to talk ball with, bless up), for the discussions, advice, collaboration, the Soedinglab coffee culture, the board game nights and the overall great lab atmosphere. I was always looking forward to coming to work.

The Soeding group OGs, especially Stefan, Jessica, Susi, and Anja, for keeping it real and making me feel included and trusted.

Sara, for the mouse collaboration, the good advice and good work together.

Wojciech, for the band, music, and early-career-researcher-but-later-than-me insights.

The Cramer group kitchen mob, the Happy Hour^{TM} team, the salsa folk, and the PBCS people - these groups are one very messy and happy Venn diagram.

The Munich COe crew, for teaching me how to behave in a social context and supplying me with amazing friends. I miss you all.

My parents, for teaching me how to live life, for supporting my choices, for giving me the foundations I needed in all things `#naturevsnurture`

Maria, for accompanying me on this journey since the very beginning; first unwittingly, then from a distance, then from inside my heart and later by my side. She is my home.

F. Myron, for the discussions, the reminders, and all the unexpected answers.

God bless.

Contents

List of Figures

List of Abbreviations

MPIBPC Max Planck Institute for Biophysical Chemistry
i.i.d. independent and identically distributed
mRNA messenger RNA
scRNA-seq single-cell RNA sequencing
UMI Unique Molecular Identifier
qPCR quantitative Polymerase Chain Reaction
TI Trajectory Inference
USD US Dollars
cDNA Complementary DNA
PCA Principal Component Analysis
ICA Independent Component Analysis
LLE Locally Linear Embedding
t-SNE t-distributed Stochastic Neighbour Embedding
UMAP Uniform Manifold Approximation and Projection
PHATE Potential of Heat-diffusion for Affinity-based Trajectory Embedding
DDRTree Discriminative Dimensionality Reduction via learning a Tree
ZINB-WaVE Zero-inflated Negative Binomial-based Wanted Variation Extraction
SLICER Selective Locally Linear Inference of Cellular Expression Relationships
PAGA PArtition-based Graph Abstraction
URD (Old Norse for "fate")
MERLoT MEthod for Reconstructing Lineage tree Topologies
EPT Elastic Principal Tree
PROSSTT PRObabilistic Simulation of ScRNA-seq Tree-like Topologies
TSCAN Tools for Single Cell ANalysis
PhD Philosophiae Doctor
HMC Hybrid Monte Carlo
CelSeq Cell Expression by Linear amplification and SEQuencing
MCMC Markov Chain Monte Carlo
ERCC External RNA Controls Consortium

1. Introduction

1.1. Transcriptome research and the dawn of single-cell transcriptomics

Multicellular organisms consist of up to $\sim 10^{14}$ cells that all carry the same genome but come in vastly different shapes and functions. This marvelous diversity arises via careful changes in expression patterns during embryogenesis and development, and is painstakingly maintained with intricate regulatory mechanisms that preserve cell type identity and therefore tissue function. Each cell expresses the subset of the genome that is relevant to its function, a combination of protein-coding messenger RNA (mRNA), transcription machinery (ribosomal and transfer RNA), as well as non-coding RNA with regulatory functions. Cellular identity has many layers, but it is fair to assume that cells with similar functions will have very similar transcriptomes.

It is no surprise that access to the transcriptome has always been in demand. Expressed sequence tags were used in the 1980s as an efficient method to determine the gene content of an organism without sequencing the entire genome, and in the 1990s reverse transcriptase qPCR could quantify the expression of selected genes. Serial analysis of gene expression allowed the capture and sequencing of random short RNA fragments, unlocking approximate quantification for the whole transcriptome. Later, microarray assays made it possible to capture the expression of thousands of transcripts simultaneously, by measuring hybridization to a specifically designed chip. In the 2000s, (bulk) RNA-sequencing took advantage of the advances offered by next-generation sequencing techniques to quantify relative expression of reverse transcribed RNA from big cell populations.

Early transcriptomics research was responsible for many impressive discoveries (differences between healthy/diseased states, esp. in cancer, pathogen response, drug resistance, host-pathogen immune interactions, role of transcript isoforms, insights in gene function, study of non-model organisms without sequenced genomes). However, scRNA-seq is the first truly high-throughput technique that can simultaneously assay the individual transcriptomes of many thousands or even millions of cells.

The feasibility of single-cell RNA sequencing (scRNA-seq) was demonstrated in 2009, covering a much higher part of the transcriptome than a single microarray experiment would [163]. The same blueprint is still followed today: single cells are isolated and lysed, the RNA is then captured, barcoded, reverse-transcribed to cDNA to increase its stability, pre-amplified via PCR and eventually sequenced. Transcripts are usually sequenced only partially, since smaller 5' or 3' fragments are sufficient to map them to the genome, but full-length transcripts contain rich information about splicing, isoforms, and single-nucleotide polymorphisms.

Key technical innovations soon allowed multiplexing the approach [71], increasing throughput from one cell to a few dozens and dropping costs (after infrastructure investments) to $\sim$10 USD/cell (for cost estimates see [207]). Using liquid handling robotics scaled the process to a few hundred cells [73], dropping costs by almost a factor 10. The addition of unique molecular identifiers (UMIs) for captured

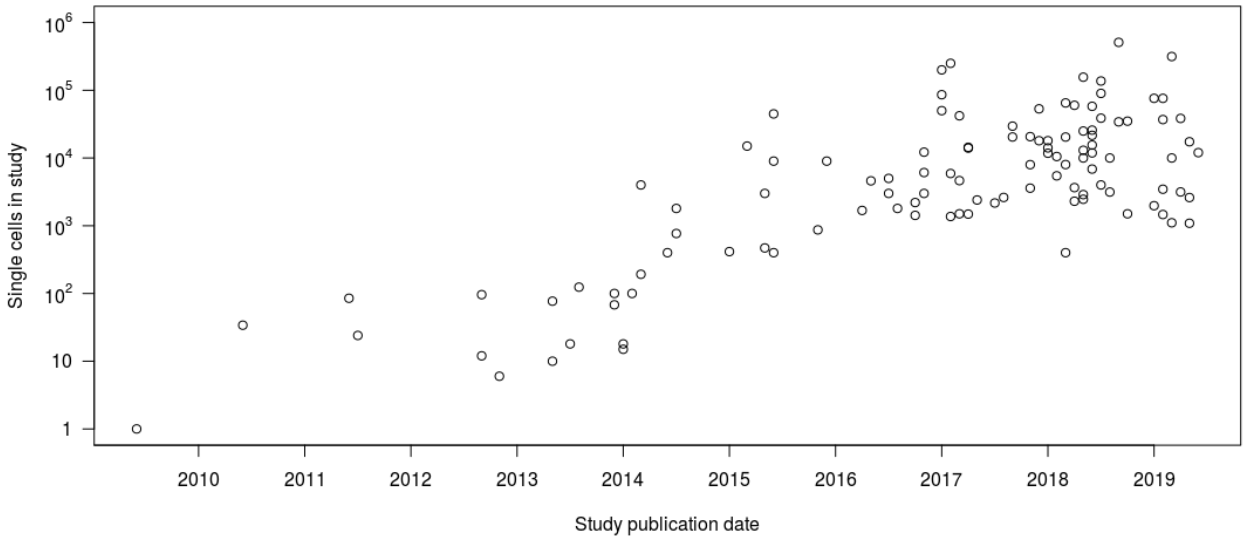

Figure 1.1.: Cell numbers reported in representative publications by publication date. Extending Fig. 1 from [162]. The number of cells available from one experiment has grown 6.5 orders of magnitude in only ten years. A full table with corresponding numbers is available in Appendix A

mRNA [72] removed qPCR amplification bias from the data. Microfluidics techniques [104, 89] truly democratized scRNA-seq, dropping per-cell costs to ∼0.1 USD/cell and increasing throughput to thousands of cells. Combinatorial indexing allowed pooling of barcoded cells, leading to more efficient chemistry and the capture of even more cells [24]. In a short ten years, experimental protocols went from one cell to half a million [203], a rapid advancement that doesn't show signs of slowing down (Fig. 1.1 and [162]).

The power of single-cell transcriptomics was impressively demonstrated in a landmark experiment by Macosko *et al.*, who performed scRNA-seq on ca. 45000 mouse retina cells [104]. They identified 39 distinct populations, most of them coinciding with known retinal cell types and some novel. One single transcriptome snapshot effectively recapitulated more than a hundred years of research in a manner of a few weeks, grouping cells not according to their appearance, behaviour, and surface markers, but rather their complete gene expression profiles, and exhibiting the possibilities scRNA-seq affords us to understand cellular states and behaviour.

These possibilities become even more tantalizing in the context of time-dependent gene expression. One or more single-cell transcriptome snapshots of a differentiating cell population (e.g. in development, wound healing, or organoids) will contain cells at various stages of differentiation. The richness of this information should, in principle, allow us to derive the lineage tree (or forest) that describes the differentiation of all cells, that maps each cell onto the lineage tree, and that can predict for all paths in that tree the detailed time courses of all gene expression levels. The first successes of single-cell transcriptomics and genomics thus herald an new era of developmental biology, in which we will be able to quantitatively measure and understand the regulatory mechanisms and networks for cellular differentiation leading to the vast, complex array of different cell types in multicellular organisms.

There is a caveat though; the starting amount of RNA in each cell is minute, and it is estimated

that only $\sim 10\%$ of the mRNAs are successfully reverse-transcribed. This means that detection of low-abundance transcripts is only possible on a population level. Moreover, the low amount of starting material also leads to high levels of technical noise, which complicates subsequent analysis and can obscure the underlying biological variation. Quality control (multiplet/broken/dead cell detection, insufficient sequencing depth, sequencing quality, mapping ratio, expression thresholds, and others) is required, something beyond the scope of this thesis. After quality control, an expression matrix $\mathbf{X} \in \mathbb{N}^{N \times G}$ of N cells and G genes is available for further analysis.

Prior to analysis, gene expression counts are usually normalized, either to a set number of total UMIs per cell or the average UMI number in the dataset, to avoid cells appearing to be similar to each other solely due to their size. Log-transforming the data is also popular, since it makes the variance of gene expression approximately normal.

1.2. Dimensionality reduction

The first step in understanding the underlying biology is often to apply dimensionality reduction techniques. This treats every cell as a vector in G-dimensional space and projects it to $D << G$ (informative) dimensions while attempting to maintain the essential characteristics of the data. If D is 2 or 3, this projection can be used for visualization. A tacit assumption is made here; namely that the projection operation that maps cells to the manifold space preserves their relationships to each other. This means, more specifically, that distances in the manifold space reflect the similarity of the expression profiles of the cells, and therefore their biological similarity. Under this assumption, and depending on the underlying biology and the granularity of the clustering, clusters of cells are considered to be part of the same grouping (loosely belonging to the same cell type).

This assumption can be extended in cases where there is a dynamic component (time, cell fate decisions): it is often assumed that the appropriate projection operation will place cells in manifold space in such a way that the underlying biological process will become clear. In this case it should be possible to arrange the cells according to their progress through the process.

Dimensionality reduction is one of the most critical steps in the analysis of scRNA-seq data, especially since it is used for visualization purposes. In the ten year lifetime of single-cell transcriptomics various dimensionality reduction methods have been adopted from different fields and new ones have been developed. As of the time of writing, the scRNA-tools database [201] contained 112 methods tagged with "dimensionality reduction". It is outside the scope of this thesis to describe every single possibility, so we will focus instead on the techniques that were most relevant for our work as well as the field overall.

All dimensionality reduction techniques take as input the original data matrix $\mathbf{X}$.

PCA

Principal Component Analysis is one of the oldest and most popular procedures for data analysis and visualization. It is an orthogonal linear transformation such that in the new coordinate system the greatest variance of the data lies on the first coordinate, the second largest on the second coordinate, and so on [80]. The first principal component of $\mathbf{X}$ is the eigenvector of $\mathbf{X}^T\mathbf{X}$ with the largest

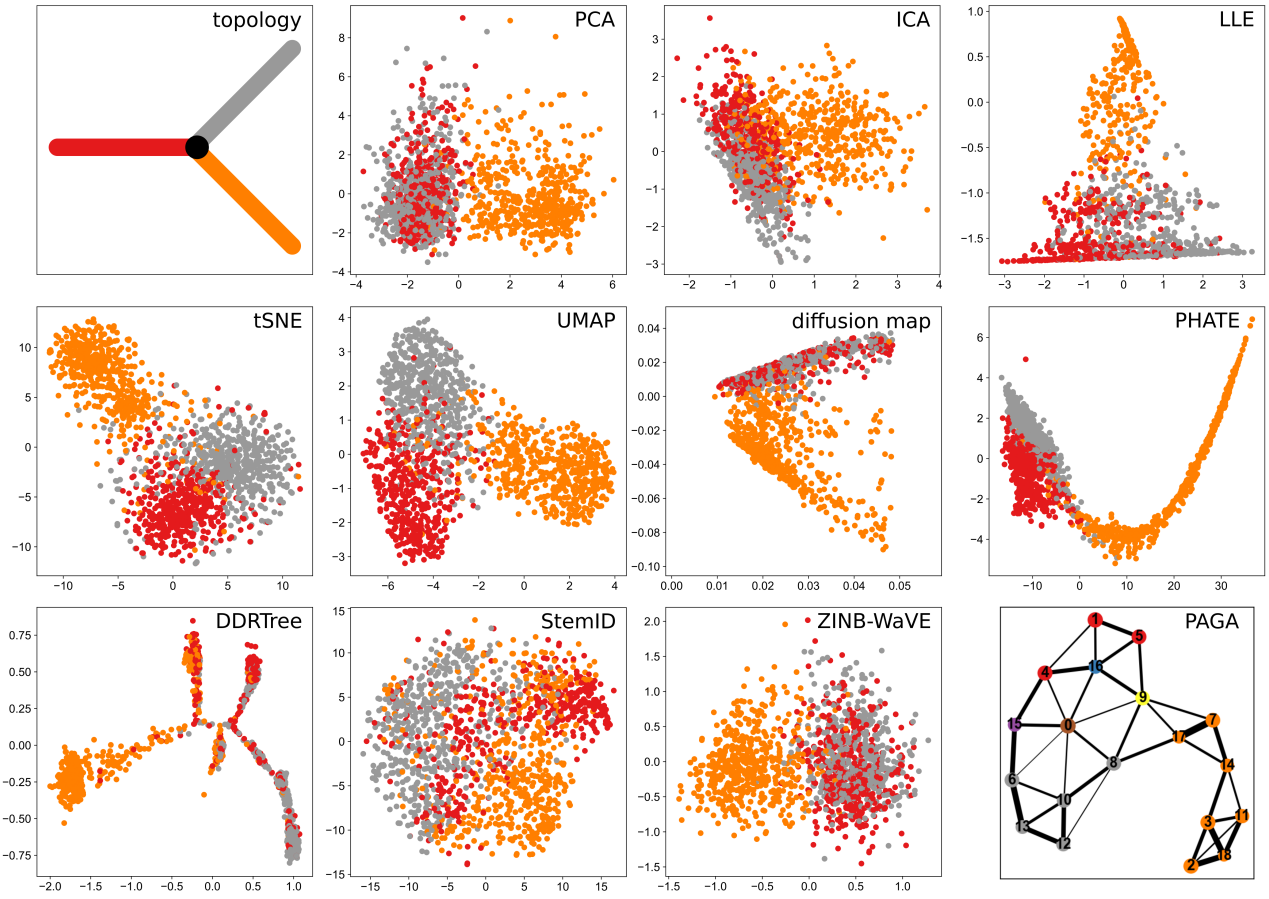

Figure 1.2.: Example of dimensionality reduction techniques applied on single bifurcation data simulated by PROSSTT [121]. The two dimensions are always the two components of the embedding, or the first two, when more are possible. The topology is indicated in the top left; the red branch bifurcates into the orange and gray one. PCA, tSNE, UMAP, diffusion map, PHATE, and PAGA are calculated using scanpy [194]. ICA and DDRTree are calculated using the Monocle package [130]. LLE is calculated using the R package "lle", StemID and ZINB-WaVE using their own R packages. The nodes with different colors in the PAGA visualization are intermediate clusters proposed by the algorithm.

eigenvalue. PCA can be very fast to compute [176] and is ubiquitously available. Its principles are easy to understand and therefore to interpret. However, the assumption that features (genes) are linearly correlated to each other is strong, and often not the case in gene expression. Additionally, PCA does not take into account the discrete nature and characteristic distributions of scRNA-seq count data.

Multiple approaches designed to improve PCA have been proposed, both within and without the single-cell field. There are multiple sparse PCA implementations [208]. ZIFA [127] implements a probabilistic PCA method to account for the presence of many zeros in scRNA-seq data. CIDR [99] imputes data on-the-fly to improve the calculation of pairwise similarity. *Harmony* iteratively corrects a PCA embedding until it integrates points from multiple datasets [91].

PCA projects $\mathbf{X}$ on N dimensions with decreasing information content.

ICA

Independent Component Analysis [70] also represents each object as a weighted sum of components that are, in some way, independent from each other. ICA models each cell x_j as an additive mixture of m components $x_j = \alpha_{j1}s_1 + \alpha_{j2}s_2 + \cdots + \alpha_{jm}s_m$, but their calculation is not based on variance considerations but rather on a more probabilistic approach. The components are posited to be statistically independent, which proves to be equivalent to demanding that their distribution is non-Gaussian. There are multiple ways to test for that (non-zero kurtosis or negentropy, minimization of mutual information, and others), and all lead to very similar results.

ICA is also a linear method, and does not take into account non-linear relationships between the expression of different genes. It projects $\mathbf{X}$ on N dimensions with decreasing information content.

LLE

PCA tries to preserve the largest (global) differences in the data by maximizing the variance explained by each component. Locally Linear Embedding [140] takes the opposite approach, by trying to preserve local similarities, and tries to recover globally non-linear structures by learning locally linear relationships.

The main idea behind LLE is that if we observe each data point and its closest neighbours in a well-sampled manifold they will lie on (or close to) a linear patch of the manifold, hence we can model each point as a linear combination of its neighbours. LLE learns the contributions W_{ij} of point j to point i by minimizing the cost function $\epsilon(W) = \sum_i |\vec{X}_i - \sum_j W_{ij}\vec{X}_j|^2$, the squared distances between each point and its reconstruction. LLE uses a fixed number of nearest neighbours for each data point, setting all other $W_{ij'}$ to 0, and demands that the contribution weights to each cell add up to 1.

This locally linear combination reflects the intrinsic properties of of each neighbourhood, and these are expected to be preserved if we project the data to a lower-dimensional manifold. LLE does that by minimizing the embedding function $\Phi(Y) = \sum_i |\vec{Y}_i - \sum_j W_{ij}\vec{Y}_j|^2$. This function minimizes locally linear reconstruction errors, just like in the original space, but now the weights are fixed and the coordinates Y_i are learned.

Due to its local averaging strategy LLE is able to capture non-linear relationships. However, in order to produce reliable results, LLE requires the manifold to be well-sampled. This is something

that is often not the case for single-cell data, either due to experimental constraints, (e.g. sampled time points are too widely spaced) or due to biological reasons (e.g. cells differentiate very fast, leaving no intermediate phenotypes to be sampled). In these cases, LLE might produce embeddings that don't reflect the biological truth of the data. Additionally, it relies on calculating Euclidean distances in the high-dimensional space, not accounting for the particular characteristics of scRNA-seq data.

LLE projects the data on a user-specified number of components, usually 2-3.

t-SNE

t-distributed Stochastic Neighbour Embedding [103] was developed to improve visualization of high-dimensional data by expanding on the idea of LLE to preserve local similarities. t-SNE does not directly compare local distances in high-dimensional and manifold space. Instead, it calculates the probability p_{ij} that two data points i, j are from the same Gaussian distribution in high-dimensional space. It then maps the points randomly to the low-dimensional space. In each algorithm iteration, t-SNE will calculate the probability q_{ij} that the manifold coordinates that correspond to i, j are from the same Gaussian distribution in low-dimensional space, and move them to minimize the Kullback-Leibler divergence between p and q.

Due to the shape of a Gaussian distribution, after a certain threshold the dissimilarity (or distance) between i and j will not matter; p_{ij} will be infinitesimal. Intuitively, t-SNE tries to keep similar objects close to each other in manifold space but will not try to keep dissimilar objects very far from each other. In this sense, while t-SNE enjoys massive popularity in the single cell community for its ability to convincingly visualize huge datasets, caution is required when interpreting a plot, as inferring cell similarity from manifold distance is not trivial. In particular, t-SNE plots often seem to display clear clustering, however this can be an artifact caused by the parametrization.

While it is generally agreed upon that scRNA-seq count data is not normally distributed, using a Gaussian kernel certainly represents an improvement compared to Euclidean distances, since it mitigates the problems caused by the curse of dimensionality. In practice, to speed up calculations, the t-SNE is often calculated on the first 50 principal components of $\mathbf{X}$ instead of the full data.

t-SNE projects the data on a user-specified number of components, usually 2-3. Embedding the data on more dimensions increases computation times considerably, since it changes the complexity of the optimization by one order of magnitude.

UMAP

While t-SNE remains a very popular visualization tool in the single cell field, recently its limitations have become more apparent too. Like other methods that are based on neighbour graphs (e.g. LLE), t-SNE uncovers local similarities at the expense of preserving the global structure. Its propensity for creating clusters often obscures continuous transitions. Additionally, t-SNE struggles when confronted with very large datasets, both in terms of representation and runtime [181]. Uniform Manifold Approximation and Projection [112] is a recently proposed dimensionality reduction/visualization algorithm that compares favorably to t-SNE [13].

Similar to LLE, UMAP breaks the manifold up in local patches, one centered around each cell. It scales its distance measure in such a way that in each patch the corresponding cell and its nearest

neighbours are uniformly distributed. This essentially creates patches of different sizes, accounting for density differences in the data space. The distances are transformed to transition probabilities between cells, merging the patches by creating a (fuzzy) graph representation of the data. UMAP then performs dimensionality reduction on $\mathbf{X}$ via spectral embedding and repeats the process, obtaining a second graph representation of the data. It then measures the distance between the two graphs using cross-entropy and changes the low-dimensional embedding with a very efficient force-directed graph drawing algorithm.

UMAP's key advantages to t-SNE are two: in terms of dimensionality reduction, the fuzzy graph representation is better at preserving the global structure of the data, while in terms of computational performance, UMAP saves a lot of time by avoiding all-against-all pairwise comparisons, instead looking at a small number of k neighbours of each cell.

UMAP projects the data on a user-specified number of components $d < D$. It does not have the same scalability issues as t-SNE.

Diffusion maps

Initially proposed by Coifman *et al.* [27], diffusion maps became a popular tool for dimensionality reduction [59, 149, 179], particularly for the analysis of single-cell differentiation data. In diffusion maps, each cell is represented as a Gaussian cloud centered at $\mathbf{x}_i \in \mathbf{X}$. The diffusion distance between two cells is the transition probability from cell i to cell j, essentially the interference between their clouds. Taking into account cell density, i.e. the number of neighbours within each cell's cloud, it is straightforward to derive a matrix $\tilde{P} \in \mathbb{R}^{N \times N}$ that describes the normalized transition probability between each pair of cells. The eigenvectors of this matrix with high eigenvalues constitute informative components that describe a low-dimensional manifold in which the process is captured.

In the calculation of diffusion maps it is critical to pick an appropriate variance σ^2 for the Gaussian that describes each cell. Lower values of σ^2 will emphasize local similarities more. As σ^2 becomes smaller the global structure of the data will be lost, as cells that were considered neighbours are now disconnected. The edge case is that every cell becomes a singleton without any neighbours. On the flip side, increasing the value of σ^2 will blur the global structure more and more, first leading to the loss of branching events and eventually grouping all cells together in a giant blob.

Diffusion maps are able to convincingly capture differentiation dynamics in real data [45]. This success though depends on three conditions. First, that the differentiation is more or less continuous, and cells transform from one type to the other by gradually changing their expression profiles. Second, that the differentiating cell population is sampled sufficiently densely that this continuity is captured in the expression profiles. Third, and most likely to be violated, is the assumption that the differentiation is an one-way street; that cells, once embarked on a certain direction, will not de-differentiate back to their original transcriptomic profile.

Diffusion maps project $\mathbf{X}$ on N dimensions with decreasing information content.

PHATE

Inspired by algorithms that find low-dimensional embeddings of words, PHATE (Potential of Heat-diffusion for Affinity-based Trajectory Embedding) was developed by Moon *et al.* [115] specifically for

dimensionality reduction of continuous data, with applications in scRNA-seq and mass cytometry but also as a general visualization tool for high-dimensional data.

Similar to diffusion maps, PHATE computes local affinities between cells and translates them to transition probabilities via a Markovian process. Instead of using the eigenstates of the transition matrix immediately, PHATE derives a heat potential representation from the transition probabilities, and uses non-metric multidimensional scaling to map them to 2 or 3 dimensions. While this doesn't preserve the distances between cells as faithfully between the high-dimensional and the low-dimensional space, the authors show that this overcomes the issues that diffusion maps have with boundary conditions. This is a valuable attribute when dealing with differentiation data, where endpoints and branchpoints are common occurrences.

PHATE is meant to be used on data where the underlying manifold is a trajectory. While its approach is in many regards an improvement to diffusion maps, PHATE emphasizes the trajectory at all costs, forcing cells to move closer to the perceived manifold. This can have unwanted side effects, like suppressing valid biological variance along the trajectory or "finding" a trajectory where none is present. Additionally, the distortion of cell-to-cell distances in the low-dimensional space means that comparisons between different trajectories are even harder to make.

PHATE UMAP projects the data on a user-specified number of components, usually 2-3.

DDRTree - Monocle 2

Monocle 2 [130] adopted reversed graph embedding [107] (RGE) to learn a tree that describes single-cell differentiation data. Instead of first projecting the data to a low-dimensional space and then mapping them to a trajectory, RGE simultaneously learns a low-dimensional graph that describes the data and a function that maps the trajectory points back to data space. This is formulated as an optimization problem: points that are connected by a graph edge should be very close in real space while the mapping function should be very faithful.

The particular RGE algorithm that Monocle2 uses is DDRTree [106], which does not require that each cell is modelled separately. Instead, it clusters cells and then performs the intensive calculations with the cluster centroids, effectively adding a kNN term to the optimization function.

DDRTree projects $\mathbf{X}$ on a user-specified number of dimensions $d < D$ with decreasing information content.

ZINB-WaVE

Zero-inflated Negative Binomial-based Wanted Variation Extraction [136] is a dimensionality reduction strategy built around the particular characteristics of scRNA-seq data. It models scRNA-seq count data using zero-inflated negative binomial distributions in order to account both for the over-abundance of zeros and the count nature of the data. ZINB-WaVE sees $\mathbf{X}$ as the sum of three matrices. The first accounts for known cell-level covariates, such as library size. The second models known gene-level covariates, such as gene length or GC-content. The third, $\mathbf{W}$, is a catch-all matrix for unknown cell-level covariates, which might be unwanted (batch effects, technical sources of noise) or have interesting biological interpretation (e.g. cell type).

It is possible to fit the hyperparameters required for this decomposition via constrained maximization

of the log-likelihood of the ZINB model. The components of the learned $\mathbf{W}$ matrix are orthogonal, and hence can be used for visualization purposes, either directly or as input to another dimensionality reduction technique, such as t-SNE.

ZINB-WaVE projects $\mathbf{X}$ on N dimensions with decreasing information content.

Other techniques

As referenced above, there are many other techniques that were adapted to scRNA-seq data or developed specifically for their analysis.

General purpose tools: Non-negative matrix factorization, with its intuitive interpretability and the possibility of analyzing Poisson-distributed data, has drawn a lot of interest [98, 196, 110, 37]. Inspired by the success of diffusion maps other techniques using spectral decompositions have emerged (e.g. CellTrails [35]). LargeVis [166] is an algorithm from the t-SNE family, exploiting a neighbour graph to project similarity structures from the data to a reduced manifold. Ensemble random projection has been suggested as an alternative [187].

Specialized algorithms for scRNA-seq data: Fully-fleged statistical approaches that model count and cell distributions carefully are still developed [3, 188]. Following the renaissance of neural networks with deep learning multiple autoencoder approaches have been proposed [6, 32, 189]. The advances in experimental protocols and the unique datasets these have created (e.g. whole embryo scRNA-seq at multiple time points) have also led to specialized software [41].

1.3. Trajectory inference and pseudotime ordering

It has been known for quite some time now that differentiation can be asynchronous (e.g. [204]), and this knowledge was reinforced when single-cell transcriptomics measurements became available [14, 153]. It became clear that correctly ordering cells according to their progress in the differentiation would allow the reconstruction of the change in gene expression during development time, potentially unlocking gene regulatory networks. This idea was formalized as the concept of *pseudotime*, proposed by Trapnell *et al.* [173] and defined as the total transcriptional change a cell undergoes as it differentiates.

Of course, in complex differentiations that included cell fate decisions, simply ordering the cells according to their transcriptional change is not enough; it is vital to recover and model this choice. When analyzing data from differentiation experiments, it is often assumed that the dimensionality reduction will preserve both the local similarities of the cells and also the global structure of the underlying manifold. Therefore, the coordinates of the cells in the reduced manifold will be arranged according to their progress through the differentiation, branching out corresponding to cell fate decisions.

If this manifold is available, a formal structure can be fitted to the density of the projected data. This structure is assumed to capture the structure of the original data; progress along this structure corresponds to pseudotime and branching events correspond to cell fate decisions. The task of constructing these structures is called *trajectory inference* and sometimes *lineage tree inference*.

There are currently more than 70 publicly available tools for trajectory inference [139, 142, 201]. Early methods usually came tied to a specific embedding (Monocle 1 with ICA [173], SLICER with

LLE [191], and others), and could only deal with specific types of trajectories (e.g. Wishbone [149] only bifurcations, [14] only linear differentiations). As more complex data became available, and it was recognized that no optimal dimensionality reduction strategy existed, later methods ([161, 123, 76] and many others) proposed more general strategies independent of embedding and became able to reconstruct complex branching topologies.

Here we will summarize a few successful methods that correspond to the state of the art.

Monocle/DDRTree

Monocle, as described above, is unique in the context of trajectory inference because it does not separate dimensionality reduction from building the trajectory. Rather, using reverse graph embedding, it learns both at once. After an initial dimensionality reduction, Monocle iteratively improves the embedding and the trajectory. In each iteration it learns a cell trajectory on the centroids of the projected data, moves the projected data points to fit the centroid trajectory, updates the centroids, and reverses the projection operation to get "improved" high-dimensional coordinates. Convergence is reached when neither the cells nor the tree need to be moved any more.

Monocle 2 is the *de facto* state of the art, being used as method for trajectory inference in many high-profile publications [24, 26, 138]. This is partly for historical reasons, since Monocle 2 predates other algorithms with similar performance, and partly because of the ease of use it provides, as it includes many downstream analysis functionalities. The fact that Monocle is further developed (with version 3 already announced) and well-maintained only add to its well-earned reputation as stable and reliable software.

Slingshot

Slingshot [161] was declared a top performing method in trajectory inference in a recent benchmark [142]. The Slingshot algorithm does not include dimensionality reduction. Rather, it expects the user to provide manifold coordinates for the cells, a clustering of these coordinates, and the cluster where the differentiation starts. Provided with these, it will first build a minimum spanning tree of the clusters to identify the global structure. Each path that connects the differentiation start to tree endpoints is then interpreted as a trajectory, and principal curves are fitted to all trajectories, providing pseudotime labels for all cells.

Slingshot (as well as TSCAN [76]) demonstrated that constructing the lineage tree using cell-to-cell connections is not robust to noise by subsampling datasets with known developmental times. However, it doesn't discard the information carried by cells entirely: when calculating cluster distances, Slingshot uses the covariance matrix of each cluster to account for cluster shape, an approach the authors consider superior to simple Euclidean distances.

RaceID/StemID/FateID

A series of related algorithms that build on each other, RaceID, StemID, and FateID represent another approach for trajectory inference.

RaceID [52] is a clustering algorithm that aims to identify rare cell types in a population. RaceID calculates pairwise Pearson correlation between all cells, converts the similarity matrix to a distance

matrix, and finds highly reproducible clusters via bootstrapped k-means clustering with a Euclidean kernel, choosing k via the gap statistic [168]. RaceID then trains a global model of the mean-variance relationship of each gene in the whole dataset and applies it to each cluster. The average gene expression per cluster is used to derive the expected variance in the cluster. Cells with at least two genes that are different beyond a statistical significance level compared to the cluster mean/variance are considered outliers and moved to separate clusters.

Version 2 of RaceID included different data sanitization, replaced k-means with k-medoids, and replaced the gap statistic with a heuristic depending on the intra-cluster dispersion. Version 3 of RaceID added alternative normalization schemes, allowed feature (gene) selection as part of the algorithm, and improved re-classification after outlier removal.

StemID [53] is an algorithm based on RaceID2 for trajectory inference. Dimensionality reduction is performed by projecting the data to the k eigenvectors of RaceID's correlation distance matrix that have positive eigenvalues. StemID assumes that if two clusters are connected (consecutive on a trajectory), they will have significantly more cells between their medoids than expected by chance. Each cell in each cluster is assigned to a neighbouring cluster by the length of its projection to the vector connecting the medoids, and the number is compared to a background calculated by random shuffling of cell positions. Clusters are considered to be connected if the enrichment of cells between them has a p-value lower than a certain threshold. The links that are kept form a graph that describes the underlying biology. Contrary to most available methods, the graph doesn't need to be acyclic.

Version 2 of StemID preserved the algorithmic steps but improved computational performance in the inference of significant links by adding an optional heuristic to replace repeated cell shuffling. Additionally, downstream analysis options were added, such as techniques to find groups of co-expressed genes.

FateID [66] walks backwards in pseudotime, trying to infer the fate bias of cells in the multipotent progenitor populations from cells within committed states of the possible lineages present in the data. These states are considered "targets", and the rest of the data is considered to precede them in terms of pseudotime. FateID will try to propagate the identity of the committed states backwards in pseudotime by examining the closest neighbours of cells with assigned cell state.

In each iteration a small number of nearest neighbours (always in terms of Pearson correlation) of each cluster are treated as the test set. Random forests are trained on the nearest neighbours of the test cells with a known state and then predict the state of the cells in the test set. Cells with a statistically significant bias towards one of the states are assigned to that state. The algorithm proceeds in this manner until all cells have been tested. The strategy for picking the test and training set ensures that the new test cells are not compared to the fully differentiated cells of the end state but rather on their pseudotime neighbours that already have a known commitment.

The fate bias gradient that leads to each committed state can be used to infer pseudotime and study the change of gene expression.

RNA Velocity

In most experimental protocols, 17-22% of the mRNAs that are captured will be unspliced. Instead of grouping these with spliced reads, La Manno *et al.* [94] use them to predict cellular state progression.

The first time derivative (rate of change) of spliced mRNA abundance (RNA velocity) is determined by the balance between the production of spliced mRNA from unspliced mRNA and the degradation rate. In gene expression space, the presence or absence of unspliced mRNA shows the direction the cell is moving in (future state of the cell, if we use the transcriptome as a proxy of cell identity) and the relative abundance of spliced/unspliced mRNA show the speed with which this change is happening. The RNA velocity of each cell can be visualized in reduced dimensions as arrows, providing directionality to the embedded cells and validating reconstructed lineage trees.

PAGA

Single-cell transcriptomics data of entire organisms [24, 128, 157, 18] has stretched the limits of conventional trajectory inference algorithms. They rely on dimensionality reduction, which struggles to accommodate the diversity present in whole-organism transcriptomes while still maintaining local similarities. Wolf *et al.* [193] proposed PAGA (PArtition-based Graph Abstraction), an algorithm that operates on different granularity layers, partitioning the data in increasingly broader groups until the desired level of resolution has been reached. PAGA uses UMAP's approximate kNN calculation strategy to obtain a graph representation of the high-dimensional data. The graph is clustered using the Louvain algorithm at the desired level of granularity. PAGA then calculates the connectivity between each cluster pair similar to StemID, by comparing the number of inter-cluster (cell-to-cell) edges to the one expected by chance.

URD

To analyze and visualize zebrafish embryonal development data from multiple timepoints, Farrell *et al.* [41] developed URD. The analysis pipeline starts with the calculation of a diffusion map of the data. By running an iterative breadth-search on the embedded points, their distance from the beginning of differentiation (blastula stage), and therefore their pseudotime is established. The user is required to define the tree endpoints, and URD performs random walks on the embedded space that start at pseudotime 0 and terminate when they reach one of the endpoints. After repeating this simulated diffusion approach, each cell can be assigned a weight for each endpoint that corresponds to how often it was found in a random walk that ended there. This collection of weights can be understood as membership to each available lineage during development, and can be visualized using a force-directed layout.

2. Reconstructing lineage trees with MERLoT

2.1. Introduction

In this manuscript we describe MERLoT [123] (MEthod for Reconstructing Lineage tree Topologies) a tool to infer complex, multi-branched trajectories in single-cell RNA-seq data. It also provides various visualization and statistical analysis options to aid in the subsequent analysis.

MERLoT consists of three modules. The scaffold tree (1) captures the structure of the underlying topology, by finding the location of differentiation endpoints (progenitor and mature populations) and approximate locations for the branchpoints (cell fate decisions). It iteratively applies Dijkstra's algorithm to find pairwise shortest paths in the projected space and builds a modified minimum spanning tree. The elastic tree (2) uses functionality provided by elastic principal trees (EPTs [48]) to connect the scaffold tree nodes with nodes that describe cell density on the reduced manifold. Finally, MERLoT embeds the tree into gene expression space (3), translating the graph nodes into "pseudocells" that recapitulate changes in gene expression throughout the differentiation process. MERLoT does not propose a new dimensionality reduction method; instead the users are encouraged to use one that suits the data in consideration.

After construction of the lineage tree MERLoT can detect differential gene expression between tree segments, visualize the change in gene expression along pseudotime, and construct gene correlation networks to explore patters of expression in different tree segments.

The scaffold tree is MERLoT's biggest strength, as it leads to robust identification of the underlying topology, but also a computational bottleneck, since it requires N shortest path searches. By performing local averaging via k-means clustering, big datasets can still be analyzed efficiently (sub-quadratic runtime) without performance loss.

We simulated 2000 single-cell differentiation experiments using PROSSTT [121] and another 400 using Splatter [200], and evaluated the performance of MERLoT, SLICER [191], Monocle 2 [130], slingshot [161], [10], and TSCAN [76]. MERLoT outperforms the state of the art in both branch assignment and pseudotime prediction for datasets with 3-12 cell fates and different cell and gene numbers. More importantly, MERLoT captures the heterogeneity present in real data (e.g. hematopoietic system reconstruction, Figs. 4 and S21).

2.1.1. Author contributions

Johannes Soeding (JS) designed the algorithm. Gonzalo R. Parra (GP) implemented it. Nikolaos Papadopoulos (NP) designed and implemented the benchmark, and contributed code to MERLoT. Laura Ahumada-Arranz performed exploratory analysis of some of the datasets visualized in the manuscript under the supervision of NP. Jakob El Kholtei tested early versions of the scaffold tree and elastic tree under supervision of GP. Noah Mottelson and Yehor Horokhovsky worked on gene-gene

correlation networks under supervision of GP. Barbara Treutlein consulted at various points during development. JS, GP, and NP jointly wrote the manuscript. GP and NP revised the manuscript for first resubmission. NP expanded the benchmark for the second resubmission and performed the extensive manuscript revision.

Detailed code contributions

In particular, NP adapted the Dijkstra algorithm from `scipy` to allow tracking of visited nodes, added functions for tree visualization (flat pie plot trees, line plots for data with known pseudotime), added the local averaging, improved packaging (e.g. docker container for scaffold tree) and enforced version control, and noticed and investigated the short-circuits produced by Splatter simulations. In the main paper NP generated Figs. 4, 5, and 6, and generated the embeddings for Fig. 2. In the supplementary material, NP generated Figs. S4-S9, S15-S18, and S21, and contributed to S14. NP created R notebooks motivating the choice of appropriate elastic tree hyperparameter values (available in `https://github.com/soedinglab/merlot/tree/master/inst/example`).

2.2. Availability

MERLoT is currently available on bioRxiv (`https://www.biorxiv.org/content/10.1101/261768v2`), and under review in Nucleic Acids Research (Oxford University Press). The attached manuscript is the latest revision, with the latest text changes marked in red color.

Published online 00 Month 2019

Nucleic Acids Research, 2019, Vol. 00, No. 00 ***1–15***
doi:10.1093/nar/gkn000

Reconstructing complex lineage trees from scRNA-seq data using MERLoT

R. Gonzalo Parra [1,2,†,*], Nikolaos Papadopoulos [1,†], Laura Ahumada-Arranz [1], Jakob El Kholtei [1], Noah Mottelson [1], Yehor Horokhovsky [1], Barbara Treutlein [3], and Johannes Soeding [1,** †]

[1]Quantitative and Computational Biology Group, Max Planck Institute for Biophysical Chemistry, Am Fassberg 11, 37077, Goettingen, Germany, [2]Genome Biology Unit, European Molecular Biology Laboratory, Meyerhofstraße 1, 69117, Heidelberg, Germany, [3]Department of Evolutionary Genetics, Max Planck Institute for Evolutionary Anthropology, Deutscher Platz 6, 04103, Leipzig, Germany

Received Month day, 2018; Revised; Accepted

ABSTRACT

Advances in single-cell transcriptomics techniques are revolutionizing studies of cellular differentiation and heterogeneity. It has become possible to track the trajectory of thousands of genes across the cellular lineage trees that represent the temporal emergence of cell types during dynamic processes. However, reconstruction of cellular lineage trees with more than a few cell fates has proved challenging. We present MERLoT (**`https://github.com/soedinglab/merlot`**), a flexible and user-friendly tool to reconstruct complex lineage trees from single-cell transcriptomics data. It can impute temporal gene expression profiles along the reconstructed tree. We show MERLoT's capabilities on various real cases and hundreds of simulated datasets.

INTRODUCTION

BACKGROUND

Recent advances in single-cell sequencing techniques (1, 2, 3) permit to measure the expression profiles of tens of thousands of cells making ambitious projects like the single-cell transcriptional profiling of a whole organism (4) or the Human Cell Atlas (5) possible. These efforts will better characterize the different cell types in multicellular organisms and their lineage relationships (6). The advances also put within reach the question of how single cells develop into tissues, organs or entire organisms, one of the most fascinating and ambitious goals in biology that would also have wide-ranging consequences for the study of many human diseases.

It is critical to develop methods that can reliably reconstruct cellular lineage trees that reflect the process by which mature cell types differentiate from progenitor cells. This is challenging due to the inherently high statistical noise levels in single cell transcriptomes, the high-dimensionality of gene expression space, and the strong non-linearities among gene interactions due to multiple transcriptional programs running in parallel for specifying the different cell type identities (6).

Different methods have been developed in the last years for inferring single-cell trajectories (7, 8). Most of these methods first apply a manifold embedding in order to reduce the dimensionality of the problem and then implement various strategies for reconstructing the trajectory structure on it. Some tools are intended for linear topologies, while others aim to resolve bifurcations, multifurcations, or even complex trees with many internal branchpoints. The latter case has proven very challenging, and there is much room for improvement. Here we present MERLoT (MEthod for Reconstructing Lineage tree Topologies), a tool that can reconstruct highly complex tree topologies containing multiple cell types and bifurcations.

MERLoT uses a low-dimensional embedding to reconstruct the cellular lineage tree topology and then maps this topology to the original high-dimensional expression space. Different manifolds have been shown to be useful for the reconstruction of different lineage trees. MERLoT implements diffusion maps (9) as produced by the Destiny package (10) as the default method for dimensionality reduction. However, users can provide MERLoT with any low-dimensional space coordinates to perform the tree reconstruction.

MERLoT explicitly models the tree structure, defining its endpoints, branchpoints, and locating a set of support nodes between these that act as local neighbourhoods for cells. This model-based strategy gives insights into the temporal order of branching and the emergence of intermediary cell types. Once the lineage tree has been reconstructed in the low

*To whom correspondence should be addressed soeding@mpibpc.mpg.de, gonzalo.parra@embl.de
†These authors contributed equally to this work.

dimensional space, MERLoT is able to embed it back to the high dimensional gene expression space. The support nodes play a two-fold role in this step: they integrate the gene expression information of the cells assigned to them, and they inform the gene expression profiles of nearby support nodes. This reduces the overall noise levels, interpolates gene expression values for lowly sampled regions of the lineage tree and imputes missing expression values.

We show MERLoT's performance on several real datasets, using different manifold embeddings, and on hundreds of simulated datasets. We generated a total of 2000 synthetic datasets with PROSSTT (11), divided into subsets of 100 simulations containing from 1 to 10 bifurcations. To the best of our knowledge, this benchmark datasets is the largest and most complete one up to date (available at `http://wwwuser.gwdg.de/~compbiol/merlot/`). We show that MERLoT outperforms other methods by producing a better classification of cells to the different branches that constitute the lineage trees. This is crucial when studying the progression of gene expression along the different trajectories in the tree, since a sub-optimal classification of cells mixing different cell types together leads to inaccurate imputation of gene expression time courses and impairs downstream analysis (3).

We repeated the benchmark with simulations generated by another tool, Splatter (12). For more information, details about method performance, and divergence analysis of the simulations, please refer to the supplementary material.

MERLoT is implemented as an R package and publicly available at `https://github.com/soedinglab/merlot`. MERLoT allows users to easily retrieve subpopulations of cells that belong to specific branches or belong to specific paths along the tree. It can also calculate pseudotime assignments, impute pseudotemporal gene expression profiles, or find genes that are differentially expressed on different tree segments.

MATERIALS AND METHODS

Terminology

We model cellular lineage trees such as the ones that result from single-cell snapshots of differentiating populations with trees as defined in graph theory, i.e. undirected graphs in which any two vertices are connected by exactly one path (13). In the context of an Elastic Principal Tree (EPT), each node is referred as a support node in the lower dimensional space. When embedding the EPT into the gene expression space $\mathbb{R}^G$, support node v_n is referred to as *pseudocell* n, since it contains the imputed gene expression values based on the cells assigned to them.

For illustration purposes, consider an experiment where quiescent progenitor cells A are given a differentiation signal, mature for a time period and then either differentiate to specific progenitors B or become fully differentiated cells C. Nodes with exactly one neighbour are called *endpoints*. They correspond to ending or starting points of the process captured in the experiment (A, B, and C). Nodes with more than two neighbours are named *branchpoints*. In this example it will be the node where the maturation ends and the cell fate decision is made. Paths between endpoints and branchpoints or between two branchpoints are named *branches*. The collection of endpoints, branchpoints, and their connectivity is the *topology* of the tree, and can be described by conventions such as the Newick tree format (`http://evolution.genetics.washington.edu/phylip/newick_doc.html`). We refer to trees with many branch points as having *complex* topologies.

Lineage Tree Reconstruction by MERLoT

Given an expression matrix with N cells as rows and G genes as columns, a manifold embedding technique can project the data onto a number of informative dimensions $D << G$. Since many dimensionality reduction techniques project the data onto mutually orthogonal dimensions, two or three dimensions often do not capture the true topology of the data. In practice, we have found that for topologies with N branches we needed $N+1$ dimensions for optimal results. Determining the correct number of dimensions to use is not trivial, and using more dimensions than needed might introduce undesired noise.

MERLoT first calculates a Scaffold Tree on the reduced manifold by locating endpoints, branchpoints and their connectivity. This is refined and smoothed by fitting an Elastic Principal Tree (EPT) on the Scaffold Tree. Finally the EPT is embedded into the high-dimensional gene expression space and pseudotime values are assigned to the cells.

Direct application of the EPT algorithm on the reduced coordinates did not yield optimal results, as it often returns trees that are manifestly far from the correct topology, e.g. with wrong number of endpoints or grossly misplaced branchpoints. (see Fig. S19C where the ICM branch, for the Guo dataset, is completely lost). This happens because, by default, the EPT algorithm tries to fit a tree structure that satisfies global energetic constraints. In this context, collapsing bifurcations leads to lower energy structures. Initialization with the scaffold tree helps to avoid this pitfall.

Scaffold Tree Reconstruction

We calculate the shortest paths p_{ij} between all pairs of cells i and j that minimize the squared Euclidean distance d_{ij}^2 (using the distance d_{ij} would only discover the edge i,j and not a longer path). We use a modified version of the `csgraph` module from the `scipy` (`https://www.scipy.org/`) library, available at `https://github.com/soedinglab/csgraph_mod`.

The shortest path p_{kl} that maximizes the number of cells S_{kl} (or the longest total euclidean distance in case of ties) is added to the tree T and the cells k,l added to the set of endpoints of the lineage tree $\mathcal{E}$. Additional endpoints n are iteratively added to the tree by selecting the shortest paths $p_{ni}, i \in T$ that maximize $s_{\mathcal{E}}(n)$, the number of cells added to T:

$$s_{\mathcal{E}}(n) := 0.5 \times \min\{S_{kn} + S_{nl} - S_{kl} : k,l \in \mathcal{E}\}. \quad \textbf{(1)}$$

In "auto" mode every time a new endpoint is proposed we evaluate if $\max\{s_{\mathcal{E}}(n',) : 1 \leq n' \leq N, n' \notin \mathcal{E}\} > \sqrt{N}$ holds true. Otherwise, we calculate the branchpoints and tree connectivity for the endpoints in $\mathcal{E}$, including n, using the methodology explained in the next subsection. After this, all cells are mapped to their closest branch. If the branch added by the selected n endpoint contains more than `MinBranchCells`$= \sqrt{N}$ cells mapped to it, the branch is kept in the tree scaffold structure and the endpoints search is repeated. Otherwise, the endpoint search terminates and n is discarded as endpoint. The `MinBranchCells` threshold can be modified by the user. Alternatively, instead of using a stop criterion, users can set the number of endpoints that are aimed to be found (fixed mode) regardless of the branch lengths.

After locating all endpoints we use the Neighbour Joining (NJ) criterion (14) in order to derive a tree (Fig. S2). Let $\mathcal{V}$ be the set of yet unprocessed endpoint and branchpoint nodes of the tree. We initialize $\mathcal{V} \leftarrow \mathcal{E}$ with the endpoint set $\mathcal{E}$.

We pick the two nodes k, l in $\mathcal{V}$ that are guaranteed to be next neighbours and therefore can be linked via a single branchpoint by minimizing the neighbour-joining distance d_{kl}^{NJ}:

$$d_{kl}^{\mathrm{NJ}} := S_{kl} - \frac{1}{|\mathcal{V}-2|} \sum_{m \in \mathcal{V}} (S_{mk} + S_{ml}). \quad \textbf{(2)}$$

The branchpoint cell m between k,l has a minimal distance from k and l as well minimal average distance from all other nodes in $\mathcal{V}$:

$$m = \arg_m \min \left\{ S_{km} + S_{lm} + \frac{\sum_{n \in \mathcal{V} \setminus \{k,l\}} S_{nm}}{|\mathcal{V}-2|} \right\}. \quad \textbf{(3)}$$

In $\mathcal{V}$ k and l are replaced by m, while the edges $l-m$ and $m-k$ are added to the tree (Fig. S2). This is repeated until $|\mathcal{V}| = 2$, where the remaining nodes trivially fulfill the criterion and can be joined. Note that the same cell can be detected more than once as a branchpoint.

Local averaging mode: Dijkstra's shortest path algorithm has, in the `scipy` implementation that MERLoT uses, a time complexity of $\mathcal{O}(N(Nk + N\log(N)))$ where N is the number of nodes (cells) and k the average number of connected edges per node (cell). Since MERLoT does not impose a cut-off on k, it is equal to N, and the runtime becomes $\mathcal{O}(N^3 + N^2\log(N))$. This means that a linear increase in cell number leads to more than a cubic increase in runtime.

To speed up the calculation of the scaffold tree, we implemented a local averaging strategy. Given a number of centroids we cluster the manifold coordinates of the cells with k-nearest neighbours (knn) (15) and subsequently calculate the scaffold tree on the cluster centroids. This knn-reduced scaffold tree is then used as input to the EPT, which returns a knn-reduced elastic tree. This can then be inflated with the original manifold coordinates. For the "deep" benchmark (see Section) we reduced coordinates to $4\sqrt{N}$ cells, for an effective runtime of $\mathcal{O}(64N\sqrt{N} + 2N\log(4N))$.

Apart from a massive speed-up, the local averaging strategy also made the quality of the elastic trees less dependent on the choice of elasticity hyperparameters. Local averaging adds little value for small datasets, as averaging over very limited samples only worsens the signal-to-noise ratio. We recommend that MERLoT should be used with local averaging for large datasets (more than a few thousand cells).

Elastic Principal Tree in the Low Dimensional Manifold

To produce smoother, more homogeneously interpolated lineage trees MERLoT uses the Elastic Principal Trees (EPT) algorithm (16, 17) as implemented in the ElPiGraph.R module (`https://github.com/Albluca/ElPiGraph.R` and described on arXiv: `https://arxiv.org/abs/1804.07580`). The EPT algorithm is used to approximate the distribution of cells in a given space with a tree structure composed of k nodes. Direct application of the EPT algorithm is unstable as it often returns trees that are manifestly far from the global optimum (e.g. wrong number of endpoints or grossly misplaced branchpoints). This can be observed in the tree reconstructions that were performed using the non-initialized EPT using ElPigraph (Fig. S19). The recovered tree topologies have more (small) branches than MERLoT reconstructions, while neighboring bifurcations found by our method get collapsed. We use the `computeElasticPrincipalCurve` function from ElPiGraph, that we initialize with the coordinates of the scaffold tree endpoints and branchpoints and the edges among them. This function will not change the scaffold tree topology used as an initialization point but add nodes to the EPT by iterative bisection of edges until it reaches the specified number of k support nodes. As a result a smoothed version of the scaffold tree is obtained.

We performed a grid search around the default EPT hyperparameter values by visually examining reconstructed EPTs with $k = 100$ support nodes on the

datasets shown in Fig. 2. We obtained $\mu_0 = 0.0025$ and $\lambda_0 = 0.8 \cdot 10^{-9}$, values that have held up well for simulated datasets (see below). For different values of k we adjust according to $\mu = (k-1)\mu_0$ and $\lambda = (k-2)^3\lambda_0$. All reconstructions in our benchmark were performed with the standard function using $k = 100$.

For some particular topologies μ and λ might need to be tuned in order to produce optimal results, in particular if k is increased a lot. Alternatively, MERLoT can bisect the edges in a given EPT, by additional nodes producing a new EPT with almost $2k$ support nodes. Note that these are special cases. An in-depth discussion of elasticity hyperparameters can be found in the Supplementary Material.

Elastic Principal Tree Embedding into the Gene Expression Space

First, cells are assigned to their closest support node according to euclidean distance in manifold space. Their average expression profile is used to initialize the "expression profile" of the node. Nodes without cells assigned to them are initialized with a null vector. By constructing such "pseudocells", we translate the positions of the support nodes in the low-dimensional manifold space to approximate positions in the full gene expression space.

Finally, the EPT algorithm is initialized with the average expression profiles and a list of edges representing their connectivity in the low dimensional EPT to calculate an EPT in the high-dimensional gene expression space.

Pseudotime Assignment

Pseudotime is a quantitative measure of the progress of a cell through a biological process (18). Given the reconstruction of a lineage tree by MERLoT, cells can be assigned pseudotime values as a function of the number of edges along the structure that separate them from the initial point of the process. MERLoT automatically sets the initial pseudotime, t_0, to one of the first two detected endpoints. Users can also set t_0 to any endpoint, branchpoint or to any individual cell. In the latter case, the closest node to that cell will be assigned as t_0 and the pseudotime values for the other nodes will be assigned as before. After a pseudotime value is assigned to each support node, cells will take the pseudotime value from their closest node in the tree. Alternatively, cells can be projected to the edge that connects their two nearest support nodes and thereby receive continuous pseudotime labels. MERLoT can calculate pseudotime in both the low-dimensional manifold space and in the high-dimensional gene expression space.

Benchmark on Synthetic Datasets

We evaluated the performance of MERLoT and other lineage tree reconstruction methods on synthetic data. We produced two simulation sets with PROSSTT (11) and one more with Splatter (12). Each set contains 1000 (10×100) datasets with 1-10 bifurcations (3-12 endpoints) each. In the first PROSSTT set ("lean") we sampled for each lineage tree 50 cells from every branch, while in the second ("deep") we used the same lineage trees but sampled 500 cells from each branch. In the Splatter set we sampled 100 cells from each branch.

PROSSTT generates a simulated scRNA-seq dataset in four steps: (1) it generates a tree (number and length of branches, connectivity), (2) it simulates average gene expression levels $\mu_g(t,b)$ (pseudotime t, branch b), (3) it samples points in the tree (t, b) (4) it retrieves $\mu_g(t,b)$ for each sampled point and draws UMI counts from a negative binomial distribution.

We provide the scripts used to create the simulations (`https://github.com/soedinglab/merlot-scripts`) as well as the simulations themselves (`http://wwwuser.gwdg.de/~compbiol/merlot/`). Detailed information about the simulation procedure and parameter values can be found in the Supplementary Material.

Splatter takes a slightly different approach to the simulation of lineage trees ("paths" in the terminology of Splatter). It is a software primarily designed to simulate populations of cells with differential expression between them. In order to simulate a lineage tree, it simulates populations with differential expression at each successive waypoint of the lineage tree (between start and first branchpoint, between successive branchpoints, between branchpoints and endpoints), and then simulates how gene expression changes from one waypoint to the other.

For parameter selection, we kept default parameters as far as the count model and the generation of average gene expression values were concerned (parameters controlling mean, library size, expression outlier, biological coefficient of variation). The rest of the simulation parameters were picked to mirror those in the PROSSTT simulations (see Supplementary Material).

Datasets

Myeloid progenitors differentiation (Paul, et al. Cell, 2015 (19)) This dataset was produced applying massively parallel single-cell RNA-seq (MARS-seq) which uses unique molecular identifiers (UMIs). After quality control and selection of informative genes the expression matrix contains 2730 cells and 3459 genes. Originally the authors reported 3461 informative genes with some of them being incorrectly formatted as dates, e.g 5-Mar, 4-Sep. We were able to correct the IDs of most of them to valid GeneIDs except two (IDs: 7-Sep and 2-Mar) which were excluded from the analysis (`https://github.com/soedinglab/merlot/tree/master/inst/example/ExamplePaul2015.R`).

Mouse zygote to blastocyst (Guo et. al, Developmental Cell, 2010 (20)) The dataset was produced by the Biomark RT-qPCR system and contains Ct values for 48 genes measured in 442 mouse embryonic stem cells at 7 different developmental time points, from

the zygote to blastocyst (20). The data was cleaned and normalized by following the vignette from the Destiny package. A total number of 428 cells and 48 genes were kept in the final expression matrix. A diffusion map was calculated using Destiny and the first three diffusion coordinates were used to calculate the lineage tree. We rotated the cells and tree nodes coordinates around the first axis in order to produce a two-dimensional representation of the data and improve visualization (see Fig. 2B, Fig. 2E, and `https://github.com/soedinglab/merlot/tree/master/inst/example/ExampleGuo2010.R`).

Haematopoietic Stem and Progenitor Cells (HSPCs) (Velten et. al, Nature Cell Biology, 2017 (21)) The scRNA-seq data was generated samples taken from two donor individuals (smart-seq2.HSC for individual 1 and QUARTZ-seq for individual 2), with all findings systematically compared between them. We followed the vignette for the STEMNET software available as part of its R package and obtained the normalized data, the cell types labels, and the STEMNET coordinates (see Fig. 2C, Fig. 2F, and `https://github.com/soedinglab/merlot/tree/master/inst/example/ExampleVelten2017.R`).

Differentially expressed genes detection After a linear tree reconstruction has been performed, MERLoT can easily find groups of genes being differentially expressed among different groups of cells. If two groups of cells are provided, e.g cells assigned to two branches in the tree (Fig. 3C), MERLoT performs a Kruskal-Wallis rank sum test (see Suppl. Note 4 for details) to evaluate which genes in the full expression matrix are differentially expressed on them. If a single subpopulation of cells is provided, the comparison is made against the rest of cells in the data. The entire list of genes is given as output, ordered by the test p-values results. Also, e-values are provided by multiplying the p-values by the number of G genes being tested.

Correlation Network Reconstruction We performed a Gene Correlation Network (GCN) reconstruction for the fibroblasts to neurons transdifferentiation dataset from the Treutlein group (22). We reconstructed the lineage tree, reconstructed the GCN, performed gene clustering and differential gene expression analysis. The script for performing this analysis is available at `https://github.com/soedinglab/merlot/tree/master/inst/example/`.

Network construction: Given the expression profile of a gene in all cells (non-imputed values) or in the tree support nodes (imputed values) we create a matrix that contains the pairwise Pearson's correlation coefficient between all pairs of genes. Given this matrix we can use the R package *igraph* (`http://igraph.org/r/`) by defining a threshold to decide which Pearson's correlation coefficients to include as weights for the edges in the graph. For the correlation measures and cut-offs used, please refer to Supplementary Figures S10-S14.

Cluster Analysis: We clustered the genes on the GCN by applying the *walktrap* algorithm as implemented in the *igraph* package. This algorithm partitions a graph into densely connected parts of a network (modules) by exploiting the fact that short random walks tend to stay within a module (23). Any module with less than a specified lower boundary of nodes is dissolved and those genes are considered as unclustered. We set the minimum number of genes to 3.

Gene Ontology term enrichment analysis: To help determine the function of network defined clusters, a module for Gene Ontology (GO) term enrichment was implemented. For this purpose the R package topGO was used. Enrichment was computed by the Fisher exact test with the gene set of the data set as the gene universe. Gene ontology tables per cluster were retrieved and representative keyboards were selected by cluster.

Differentially expressed genes: We detected the differentially expressed genes at every branch of the reconstructed lineage tree using the `branch_differential_expression` function from MERLoT. All genes are ranked according to the e-value obtained from the Kruskal-Wallis rank sum test, applied to test whether they are differentially expressed or not. Genes with e-val $<10^{-3}$ are considered to be differentially expressed. Genes are colored according to the mean difference in expression between the two compared sets of cells (i.e. selected branch and rest of the tree for each case). Upregulated genes are shown in shades of red and downregulated genes in shades of blue.. Intensity corresponds to log fold change of gene expression. Genes that are not significantly differentially expressed are colored in black.

RESULTS AND DISCUSSION

MERLoT's workflow for lineage trees reconstruction and gene expression imputation

Given the matrix of expression values for all cells (Fig. 1A), MERLoT reconstructs lineage trees according to the following steps (for details see Methods): First, MERLoT applies a dimensionality reduction method to map the high-dimensional expression vectors of cells to a low-dimensional space. Users can replace the default, diffusion maps, with the method of their choice. Second, MERLoT calculates a scaffold tree in the low-dimensional space combining the Dijkstra's shortest path (24) and Neighbour Joining (14) algorithms to define the location of endpoints, branchpoints and their connectivity (Fig. 1B). (View Suppl. Note 5 for a pseudocode explanation). The scaffold tree is used as initialization for an Elastic Principal Tree (EPT) (16). The EPT smooths the scaffold via an optimization procedure that places a user-defined number of support

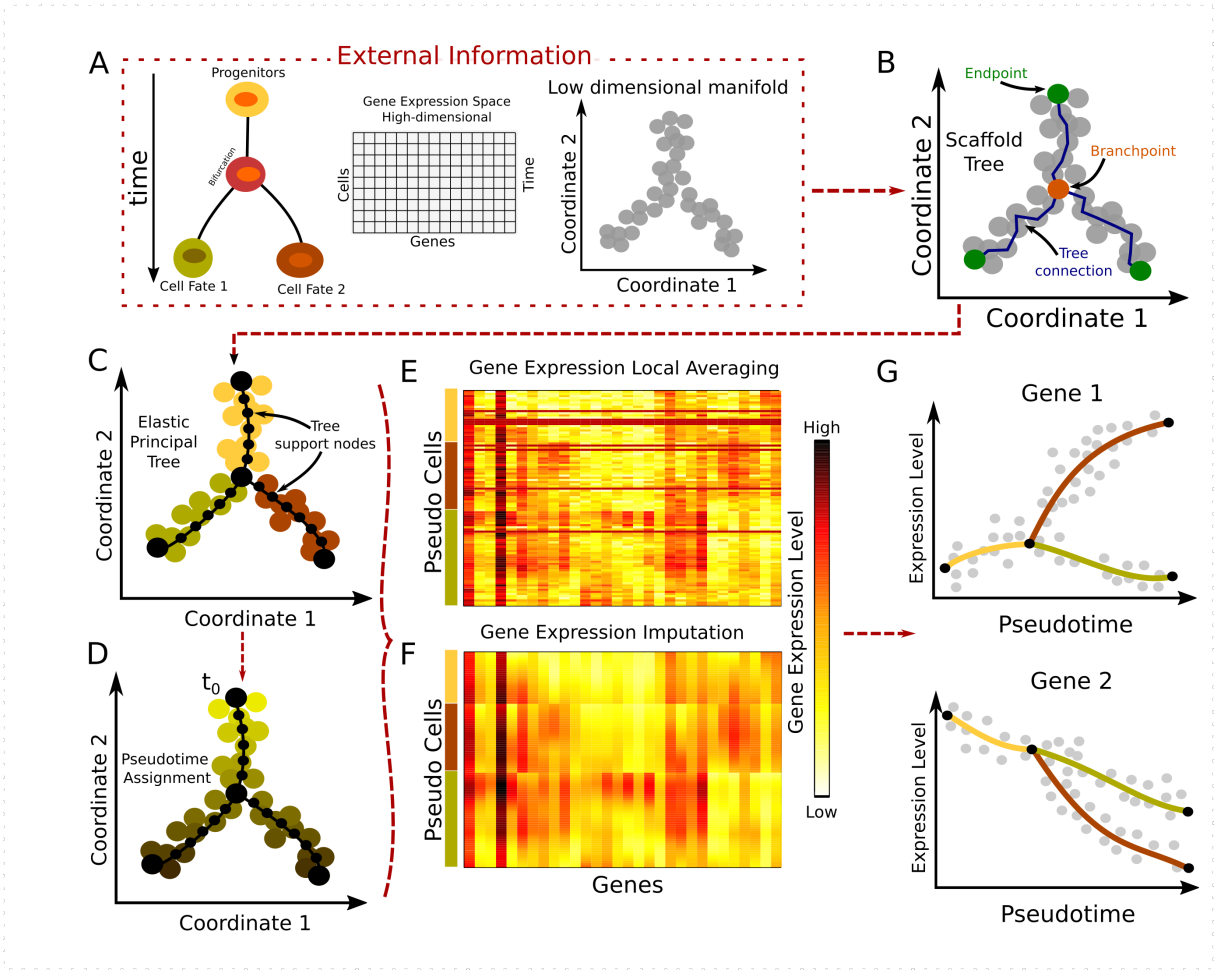

Figure 1. MERLoT's workflow: (A) Input to MERLoT is a gene expression matrix sampled from a dynamic process in which several cell types are present. MERLoT uses diffusion maps to reduce the dimensionality of the expression vectors for each cell to a few components (typically between 2 and 20). Users can provide any low-dimensional manifold set of coordinates to MERLoT as input. **(B)** A scaffold tree is calculated given the low-dimensional manifold coordinates. **(C)** The scaffold tree is used to initialize a principal elastic tree, composed of k support nodes (default: 100), on which cells are assigned to the different branches of the tree. **(D)** Given a cell or tree node as the initial pseudotime t_0, pseudotime values propagate to the rest of cells/support nodes proportional to the distance along the tree that separates them from t_0. **(E)** Expression values from cells assigned to a given support node or pseudocells (see main text) are averaged to provide it with an averaged expression profile for each gene. **(F)** Gene expression values after imputation and interpolation in the gene expression space: A high-dimensional principal elastic tree is initialized with the connectivity from the low dimensional principal elastic tree plus the averaged expression values from the support nodes to impute smoothed gene gene expression data for each gene in the gene expression space. **(G)** MERLoT imputes the pseudotime-dependent expression profile of each gene along each branch in the tree. Gene expression can be visualized as a function of pseudotime.

nodes between endpoints and their corresponding branchpoints interpolating the density of cells in the low-dimensional space (Fig. 1C). Once the low-dimensional tree is optimized, an initial pseudotime t_0 is assigned to the user-specified tree root. The pseudotime of each cell is then proportional to its distance from the root along the tree structure (Fig. 1D).

To study gene expression changes along the different tree branches, MERLoT embeds the low-dimensional EPT structure into the high-dimensional gene expression space. Each tree support node in the low-dimensional space is mapped one-to-one to a tree support node in the gene expression space: We first assign each cell in the low-dimensional space to its nearest support node. Then, we initialize the corresponding support node in the gene expression space to the average gene expression level of all cells assigned to it (Fig. 1E), and we run the EPT algorithm again (Methods). In this way, we find the gene expression values of all the support nodes (Fig. 1F), which can be considered as "pseudocells", representing waypoints in the idealized cell differentiation paths and containing imputed gene expression values for their surroundings in the expression space.

The cells' pseudotime values can also be refined in this step, since cells are reassigned to support nodes in the full multi-dimensional space. By combining the imputed expression values with the pseudotime

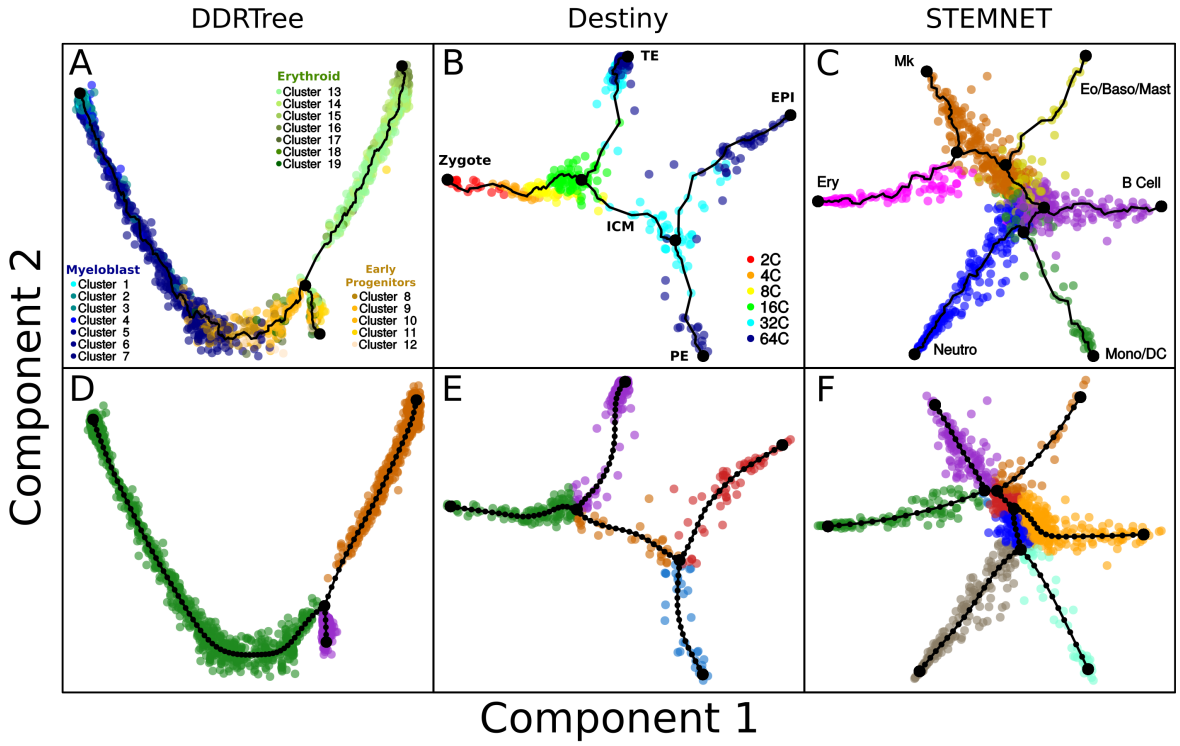

Figure 2. MERLoT's scaffold tree reconstructions: in combination with **(A)** DDRTree coordinates for analyzing the data from Paul *et al.* (19); **(B)** diffusion maps coordinates for analyzing the data from Guo *et al.* (20) (diffusion components 2 and 3 rotated around component 1 for better visualization of the data); **(C)** STEMNET coordinates for analyzing the data from Velten *et al.* (21). Cells are colored according to cell type annotations provided by the authors of each dataset. **(D, E and F)** EPT reconstructions using the scaffold trees from panels A, B, and C respectively as an initialization point. Cells are colored according to MERLoT's branch assignments.

assignments of the support nodes, MERLoT can reconstruct imputed pseudotime courses of gene expression profiles along the tree (Fig. 1G).

Applying MERLoT to real datasets

We applied MERLoT on three real datasets with different degrees of lineage tree structure complexity (details in Methods): (1) scRNA-seq data (with Unique Molecular Identifiers, UMIs) for myeloid progenitor differentiation (2730 cells, 3460 genes) (19), embedded in DDRTree coordinates (Fig. 2A, D), (2) single-cell qPCR data for zygote to blastocyst differentiation (428 cells, 48 genes) (20), embedded in a diffusion map (Fig. 2B,E), and (3) scRNA-seq data (index-omics) for haematopoietic stem and progenitor cells (1034 cells, 469 genes), using STEMNET coordinates (21). The number of endpoints of the lineage trees, found by MERLoT in "auto" mode, are consistent with the expected number of cell types that are described to be present on each of the analyzed datasets.

After the lineage tree reconstruction, each support node on the tree will be assigned a pseudotime value equal to the number of edges that separate it from the beginning of the differentiation, t_0. This can be a tree endpoint (the Zygote branch in Fig. 2B) or an internal support node (for example a node in the red branch of Fig. 2E). Each cell will be assigned the pseudotime value of its closest tree node. In Fig. 3A, pseudotime values for the zygote to blastocyst dataset are shown. Because scRNA-seq data contain a lot of technical and biological noise (6), cells with similar pseudotime values may have large variations in gene expression. MERLoT imputes denoised gene expression profiles by embedding the reconstructed lineage tree into the original gene expression space.

This model-based interpolation results in denoised pseudotime courses of gene expression for the entire tree. As an example, Fig. 3C shows the expression profiles of four genes that are differentially expressed between the epiblast (EPI, in red) and the primitive endoderm (PE, in blue) cell lineages (Methods). Note how the expression values of the pseudocells (solid lines) interpolate and smooth the noisy single-cell expression values (circles) even in regions with low cell density.

Lineage tree reconstruction in high-dimensional space

While two or three-dimensional projections of datasets are easier to visualize, MERLoT can utilize any number of informative dimensions of

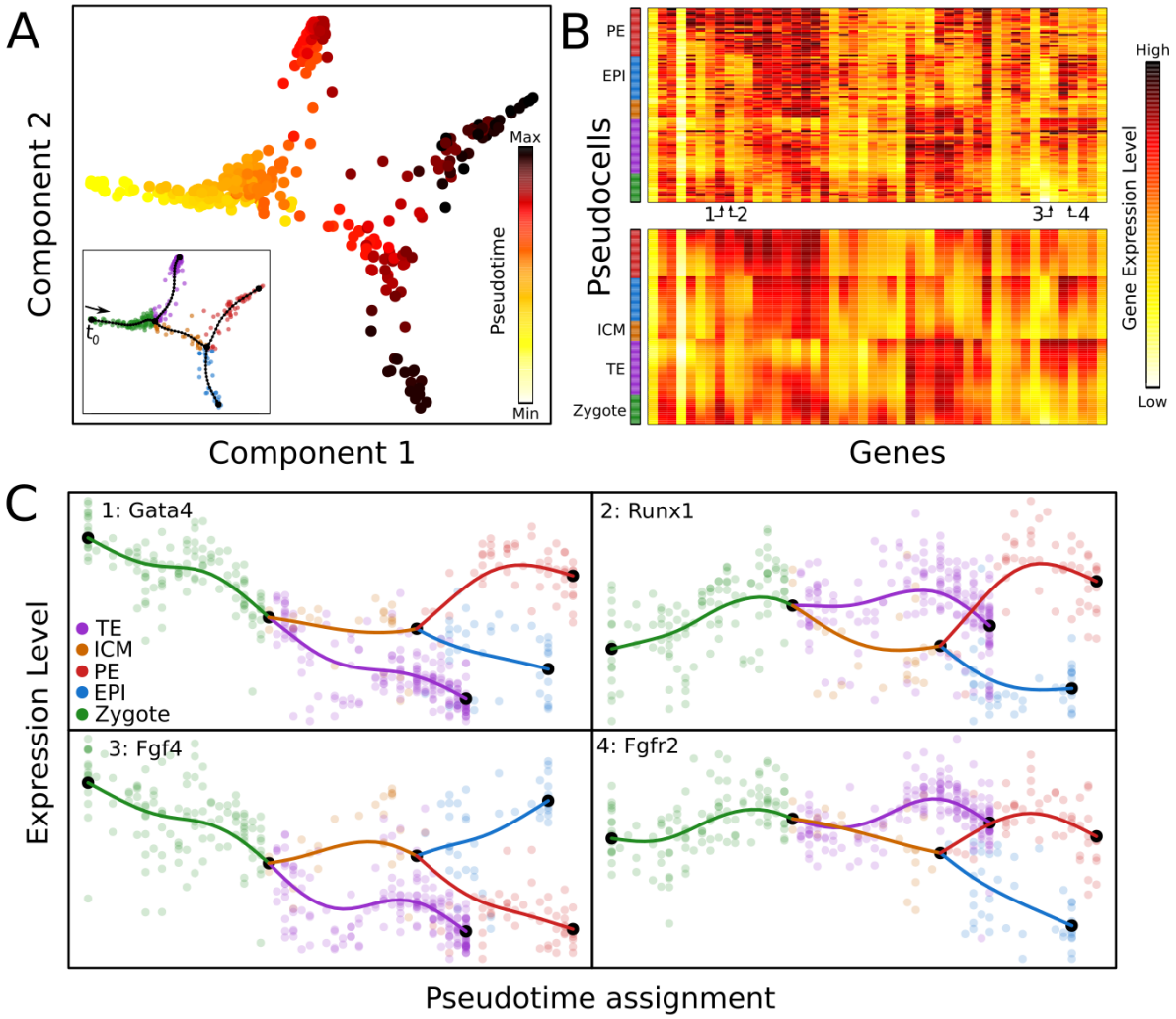

Figure 3. Pseudotime assignment and interpolation of gene expression profiles: (A) Pseudotime assignment in color code for cells from Figs 2B and 2E, taking the zygote state as t_0. **(B)** Color-coded matrix of gene expression values for tree pseudo cells (rows) times genes (columns) before (top) and after (bottom) the gene expression space interpolation using the EPT. Pseudo cells are ordered according to their pseudotime and genes were hierarchically clustered. Numbers indicate specific genes shown in panel C. The color code in the bar on the left side of the heatmaps refers to pseudo cells branch assignments. **(C)** Gene expression profiles over pseudotime for 4 genes that are differentially expressed between the EPI and PE lineages. Semi-transparent circles represent the expression values of individual cells and solid lines correspond to MERLoT's interpolations. Colors as in Fig. 2E.

the embedding space to reconstruct a lineage tree. However, visualizing hundreds or thousands of cells in multiple dimensions is challenging. MERLoT overcomes this limitation by using graph drawing techniques to project complex multidimensional EPTs into two dimensions while still displaying cell annotation and density. In Fig. S3 we show reconstructed topologies for simulations with 6, 8, 10 and 12 different cell types.

In the original Monocle2 paper (25), the authors show in Supplementary Figure 16 their analysis of the haematopoiesis dataset produced by Paul *et al.* (19). They use 10 components of the DDRTree projection to recover a topology with 5 branching points. We applied MERLoT on these coordinates and visualized the resulting tree using as annotation the cell types that Paul *et al.* assigned to the various clusters (Fig. 4; also see Suppl. Note 6: in the Supplementary Material).

Monocle2 (Fig. 4A) separates the progenitors (left-most, cyan), the erythrocytes (top, purple) and an internal branch of granulocyte/monocyte progenitors (bottom middle, brown). However, it fails to separate the megakaryocyte branch (top middle, blue) and groups dendritic cells (bottom right, red) together with basophils (green) and monocytes (orange). Furthermore, it groups neutrophils together with monocytes (right, magenta). MERLoT (Fig. 4B) achieves almost pure megakaryocyte and dendritic cell

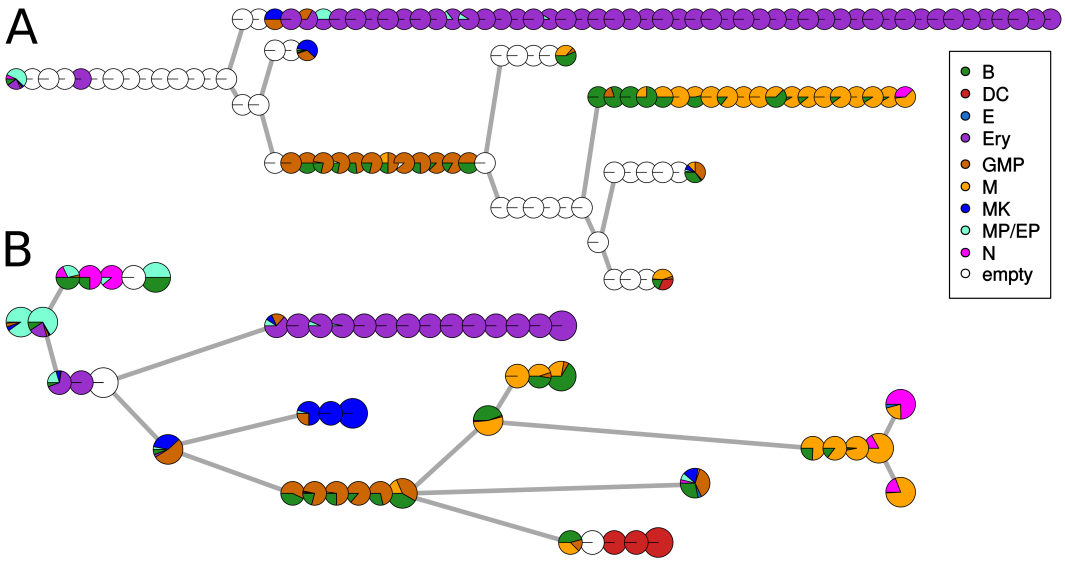

Figure 4. Reconstruction of haematopoietic system B: basophil (green); DC: dendritic cell (red); E: eosinophil (light blue); Ery: erythrocyte (purple); GMP: granulocyte and monocyte progenitor (brown); M: monocyte; MK: megakaryocyte; MP/EP: multipotent myeloid and erythroid progenitors; N: neutrophil (orange). Each pie chart represents a tree node, and the colors denote the different cell types mapped to it. The size of the pie charts does not scale with the number of cells mapped to that node (also refer to Fig. S21) **(A)** The Monocle2 reconstruction of the process. Each pie chart is one of the nodes of the minimal spanning tree. **(B)** The MERLoT reconstruction of the process. Each pie chart is a node of the EPT.

branches and separates the bulk of the neutrophils from the monocytes (also see Fig. S21C). Additionally, it separates most of the neutrophils and basophils from the bulk of the progenitor population (see Fig. S21C). Although MERLoT better resolved certain branches in the lineage tree, neither method succeeds in separating basophils from monocytes and granulocyte-monocyte progenitors, and both trees contain a branch with a mixture of almost all cell types, including pluripotent progenitors (right, middle).

Tree reconstruction performance assessment on synthetic data

In order to assess the quality of MERLoT's lineage tree reconstruction, we compared its performance to four tools with a similar approach to trajectory inference, namely unsupervised methods that produce branch assignments and assign a pseudotime to each cell: SLICER (26), Monocle2 (25), TSCAN (27), and Slingshot (28). Since our focus was on correctly predicting cell labels and cell pseudotime, we needed data from complex topologies with known intrinsic developmental time.

For this purpose we developed PROSSTT (Methods, (11)), a software that simulates scRNA-seq expression matrices with complex lineage tree structures. PROSSTT provides pseudotime and branch assignment labels for the cells, as well as branch connectivity information. Examples of diffusion maps for PROSSTT simulations and their lineage tree reconstructions with up to three bifurcations, performed by MERLoT, are shown in Fig. 5A. Additionally, we used Splatter (Methods, (12)), a suite to simulate cell populations with differential expression that can also be used to simulate branched topologies.

We generated three simulated datasets, two of 1000 simulations each by PROSSTT ("lean" and "deep") and one of 1000 simulations with Splatter. All sets contain ten subsets of 100 trees with 3 to 12 different cell fates and 1 to 10 bifurcations each. However, in the dimensionality reductions of datasets simulated by Splatter, branchpoints and endpoints often do not lie far enough from preceding tree segments. This makes it difficult for trajectory inference methods to correctly detect the tree structure, as they may connect non-adjacent tree segments, thereby creating "short-circuits" (Supplementary Material, "Divergence Analysis" and Fig. S18). Because of this reason, we only use Splatter simulations with 1-4 bifurcations for performance evaluation. The results for the higher order bifurcations and more analysis on why they were left out are shown in Supplementary material (Figs. S4, S16, and S17.)

For all simulations, we predicted the lineage trees, assigned cells to branches, and calculated cell pseudotime values using the aforementioned tools. Since MERLoT and Slingshot work on a given set of manifold dimensions provided by the user, we used them in combination with diffusion maps (provided by the Destiny package) or DDRTree coordinates (provided by Monocle2). For simplicity we will refer to these combinations as MERLoT_Destiny,

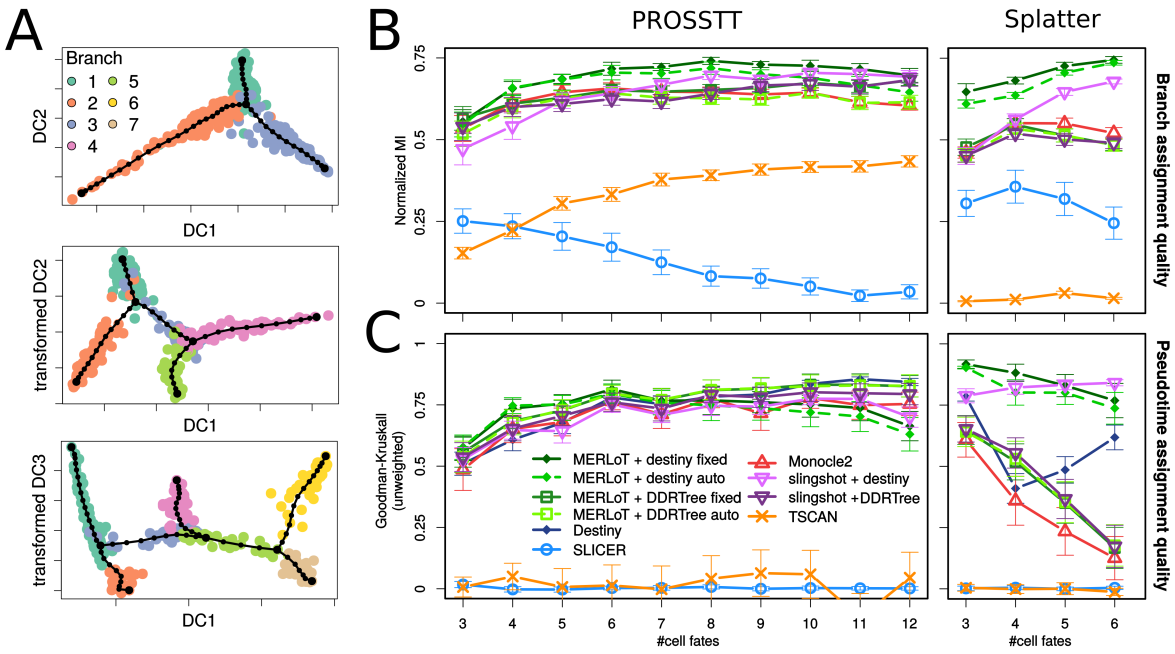

Figure 5. Simulated datasets and benchmarking: (A) Examples for diffusion map embeddings of PROSSTT simulations. From top to bottom: one, two, and three bifurcations. Cell colors the labeled branch assignments. The double bifurcation is plotted by rotating diffusion components 2 and 3 around component 1. The triple bifurcation is plotted by rotating diffusion components 3 and 4 around component 1. **(B)** Branch assignment comparison using Monocle2, SLICER, TSCAN, Slingshot and MERLoT using both PROSSTT (left) and Splatter (right) simulations. Slingshot and MERLoT are used in combination with DDRTree and diffusion map (Destiny) coordinates. **(C)** Pseudotime assignment comparison. The error bars are 95% confidence intervals assuming the prediction scores are normally distributed.

MERLoT_DDRTree, Slingshot_Destiny and Slingshot_DDRTree. Additionally, MERLoT can be used with and without providing the correct number of cell fates ("fixed" and "auto" modes). TSCAN and Slingshot do not provide a formal tree structure object, so in order to evaluate their performance we had to implement wrappers for them (see Methods).

Branch assignment quality We assessed the agreement between predicted and labeled branch assignment predictions in the simulations using the Normalized Mutual Information (NMI), since it punishes splitting and merging clusters equally (Fig. 5B). This avoids systematic advantages for methods that are biased to produce either more or less branches than the simulated ones. We also included other scoring measures (Methods and Fig. S4).

In the left side of Fig. 5B (top panel) we show the results for the "lean" benchmark set. MERLoT_Destiny consistently outperforms Monocle2 and has the best overall performance, while Slingshot_Destiny achieves comparable results for more complex topologies. SLICER scores low mainly because its recursive branch assignment function crashes for many datasets or does not finish in less than 60 minutes, especially for complex topologies. TSCAN applies dimensionality reduction based on PCA which mixes up all cell types when projected in the reduced space and hence cannot be correctly classified.

In the Splatter benchmark set (Fig. 5B, right) we only use lineage trees with up to 4 bifurcations to avoid short-circuits (Fig. S17). Still, tools with a DDRTree embedding perform worse than in the "lean" set. Slingshot_Destiny again improves for more complex topologies but MERLoT_Destiny performs better. SLICER and TSCAN still underperform for the same reasons.

With the 10-fold increase in cell numbers in the "deep" set we expected an overall increase in the performance levels of the methods. Indeed, this is the case for MERLoT, which is consistently between 5-10 percentage points better than in the original benchmark. Monocle2 performs at the same level as on the "lean" set. Slingshot_Destiny shows improvement until 5 bifurcations, where performance starts deteriorating. This happens because starting at 5 bifurcations (5500 cells) Slingshot fails to compute distances between clusters for multiple simulated datasets due to a "computational [matrix] singularity" error. This bug can be circumvented by adding small amounts of random noise to the diffusion map coordinates, in which case Slingshot's performance increases with the complexity of the topologies (dotted line).

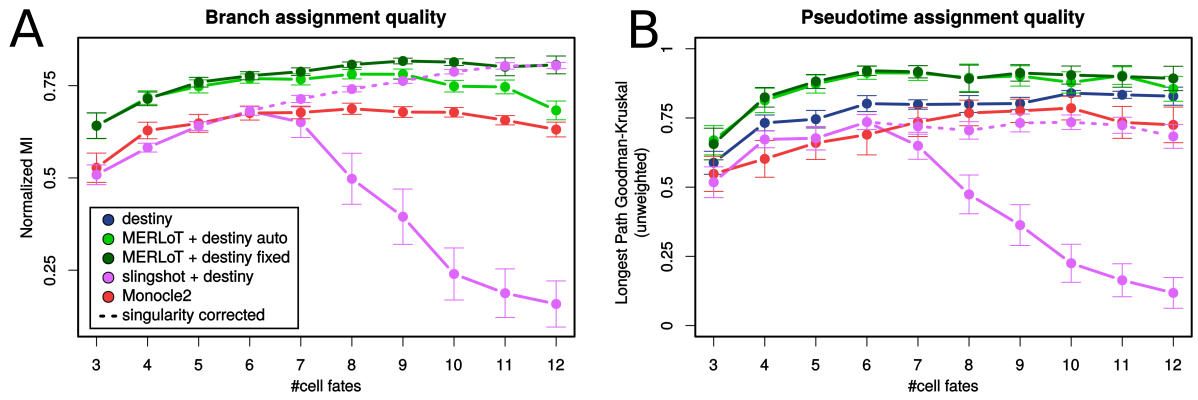

Figure 6. "Deep" benchmark set: Branch assignment (left) and pseudotime prediction (right) performance in the "deep" PROSSTT benchmark. Slingshot and MERLoT are used only in combination with diffusion map (Destiny) coordinates, since this combination performed best in the "lean" benchmark. The error bars are 95% confidence intervals assuming the prediction scores are normally distributed.

Pseudotime assignment quality In a multi-branched lineage tree, multiple trajectories exist between progenitors and differentiated cell fates. Pseudotime only assigns an ordering within each trajectory, while pseudotime values are not comparable between non-consecutive tree branches. We therefore test pseudotime orderings on the cells that belong to the longest possible trajectory in terms of pseudotime steps in every simulation. We use the Goodman-Kruskal's gamma (Figs. 5C, 6B) and other indices (see Methods and Fig. S6) as a measure of concordance between the true and predicted orderings along the longest trajectory.

In the "lean" set (5C, left), MERLoT_Destiny and Slingshot_Destiny perform better for easier topologies and are overtaken by MERLoT_DDRTree and Slingshot_DDRTree for more complex ones. MERLoT_DDRTree is overall the best method, and only gets slightly overtaken by Destiny for the most complex topologies in the benchmark. The poor performance of TSCAN and SLICER is a direct result of their poor performance at branch assignment.

In the Splatter set performance drops with the growing number of short-circuits, something that impacts Monocle2 considerably. Approaches based on diffusion maps, on the other hand, thrive.

The situation is clearer in the "deep" set. MERLoT improves dramatically compared to the "lean" set, reaching an improvement of 25 percentage points for topologies with 10 bifurcations. Destiny and Monocle2 perform at the level of the "lean" set. Slingshot_Destiny does not improve in pseudotime compared to the original benchmark, even when the computational singularities are circumvented ("singularity corrected" in Fig 6B).

Post lineage inference analysis

One of MERLoT's assets is that it can reconstruct imputed gene expression profiles to study how a gene varies along the different paths in the lineage topology. These profiles interpolate and denoise the gene expression values of cells that are assigned to equivalent pseudotimes and facilitate the study/analysis of gene expression regulation, for example via detecting modules of genes that have correlated expression profiles. By using the imputed gene expression values recovered from MERLoT downstream analysis like Gene Correlation Network (GRN) reconstruction could be improved. Building GRNs requires identifying causality for the gene-gene interactions, which exceeds the scope of this work. However, as proof of concept, we derived a Gene Correlation Network (GCN), a proxy for a GRN. We analysed the dataset in which Treutlein and coworkers studied the transdifferentiation process of fibroblasts into neurons (22). Overexpression of the proneural pioneer factor *Ascl1* causes cells to exit the cell cycle and re-focus gene expression through distinct neural transcription factors. However, later on in the process a myogenic program competes with the neural one, producing undesired myocyte-like cells and lowering the efficiency of the direct reprogramming process.

We reconstructed the lineage tree from the data, which shows a single bifurcation (Fig. 7A). Then, we embedded the tree structure into the gene expression space and recovered the imputed gene expression values for the support nodes. We calculated the Pearson's correlation coefficients between all pairs of genes using both the original gene expression values from the cells and the imputed ones from the tree support nodes. In Fig. 7B we show the Pearson's correlation coefficient distributions for imputed and non-imputed gene expression values. While the non-imputed values concentrate most values between -0.5 and 0.5, the distribution of imputed values contains two subpopulations of genes close to -1 (highly anti-correlated) and 1 (highly correlated) separating them from the rest of weakly correlated genes. In Fig. 7C and 7D we show the gene expression values of S100a6, a gene differentially expressed in myocytes, and of Ap3b2, a gene differentially expressed in neurons, respectively.

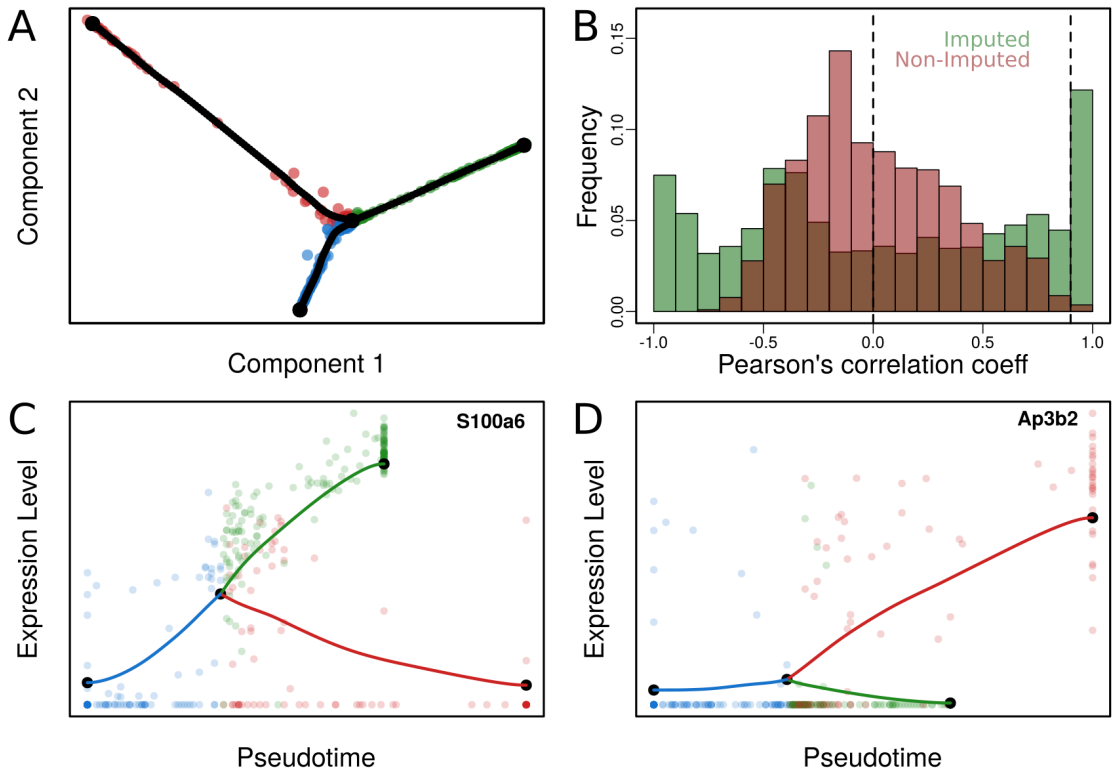

Figure 7. Lineage tree reconstruction of fibroblasts to neurons transdifferentiation: (A) Two-dimensional diffusion map embedding of cells together with the reconstructed lineage tree (tree support nodes shown in black). We observe a single bifurcated tree containing three branches corresponding to fibroblasts, neurons and myocytes. **(B)** Pairwise Pearson's correlation coefficients for gene expression profiles using imputed (green) and non-imputed (red) values. The dashed line at x=0 represents the separation between positively and negatively correlated genes. The dashed line atx=0.9 points at the threshold for reconstructing the GCN in Fig. 8. **(C)** Pseudotime gene expression profile of differentially expressed gene in myocytes. **(D)** Pseudotime gene expression profile of a differentially expressed gene in neurons.

Once all pairwise gene expression correlations have been calculated, we built a graph where nodes correspond to genes and the pairwise Pearson's correlation coefficients of imputed expression levels represent the weight of the edges connecting them. In Fig. 8 we observe the graph that results from applying a layout (see Methods) to distribute highly correlated genes (Pearson's correlation coefficients greater than 0.9) in space. On this representation, close proximity corresponds to high correlation and vice versa.

For the quality of the reconstructed GCN, concentrating on highly-correlated genes and using imputed gene expression levels is crucial. A low correlation threshold will add noise to the GCN by including false positive interactions. Additionally, the noise in the non-imputed data obscures the similarities between the time-dependent expression of genes, resulting in low correlation values (Fig. 7B). This means that using a high correlation threshold will result in a GCN with few, small connected components, while a lower threshold will inevitably lead to densely connected "hairy ball" constructs that are difficult to interpret when raw, non-imputed values are used instead (Fig. S10-S12).

After obtaining the GCN, we clustered the network (see Methods) and recovered the gene ontology (GO) terms that were enriched in each cluster (see Methods). In Fig. 8A we show the reconstructed network coloured by cluster and labeled according to the keywords that best represent their enriched GO terms. We used MERLoT's module for finding differentially expressed genes, both upregulated (red) and downregulated (blue), on each of the 3 subpopulations assigned to each branch that composes the lineage tree structure (see Methods). This was done for each branch, i.e, fibroblasts (Fig. 8B), neurons (Fig. 8C) and myocytes (Fig. 8D). Genes that are differentially upregulated in the fibroblasts branch mainly belong to the clusters enriched for GO terms related with "mitosis and proliferation" and "protein metabolism". Downregulated genes in fibroblast cells mostly belong to clusters associated with the GO terms "myogenic", "cell death and nucleic acids synthesis", and "nervous system development". We observe that for neurons, genes related to synaptic GO terms are upregulated while for myocytes the same happens for genes belonging to the myogenic cluster. Interestingly, downregulated genes on each branch

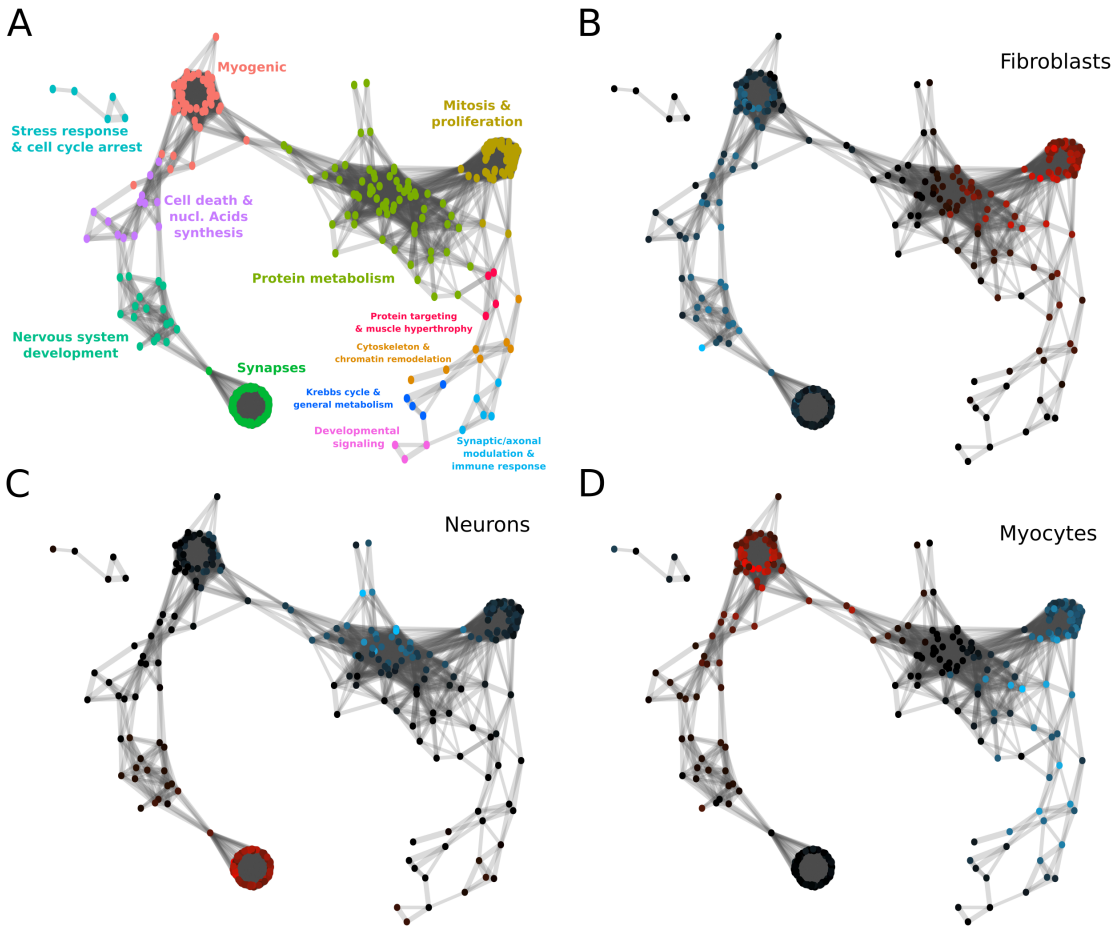

Figure 8. Reconstructed gene association network: (A) Gene clusters given the network connectivity were calculated and coloured. Enriched GO terms were retrieved for each cluster and a general label summarizing their main implications are assigned to each cluster. **(B)** Differentially expressed genes in the fibroblast branch. **(C)** Differentially expressed genes in the neurons branch. **(D)** Differentially expressed genes in the myocytes branch. Genes that are differentially expressed with an e-value $e < 10^{-3}$ are colored according to the mean difference in expression, shades of blue indicating downregulation and shades of red upregulation. Intensity corresponds to log fold change of gene expression (see color scale).

belong to interconnected clusters. While neurons downregulate genes mostly associated with protein metabolism, myocytes downregulate genes associated with mitosis and proliferation, protein targeting and muscle hypertrophy, and cytoskeleton and chromatin remodeling.

This dataset contains two mutually exclusive cell lineages, so gene expression patterns are mostly mirrored (upregulated genes in neuronal branch are downregulated in myogenic and vice versa). However when the tree topology is more complex, multiple transcriptional programs run in parallel and genes can have quite different behaviours in the different branches of the lineage tree. If GCNs are built using all cells together, spurious correlations can be recovered because of the so called Simpson's paradox (3). An analysis of this effect is shown in Suppl. Note Suppl. Note 5:, where we reconstruct GCNs for the Guo dataset (Fig. 2B)) that contains one progenitors population (zygote cells) and 3 mature cell types (TE, EPI and PE). We show that GCNs that are built using all cells don't allow a clear separation of gene markers that are specific for the EPI and PE lineages that emerge from the Inner Cell Mass (ICM) lineage.

CONCLUSION

As single-cell RNA sequencing is becoming a mainstream technology, many datasets with highly complex underlying lineage trees will need to be analyzed. Here we have presented MERLoT, a tool to reconstruct complex lineage tree topologies in a more accurate way than other methods. We show this by applying MERLoT to various published datasets, but also by extensively testing its performance on a total of 2400 simulated datasets, produced by PROSSTT

and Splatter. In this benchmark, MERLoT compares favorably to the state of the art in branch detection and pseudotime prediction using a variety of established performance indices.

Lineage tree reconstructions are not a final objective but rather a proxy to study changes in gene expression and understand the delicate regulation procedures that lead to organisms development, cellular differentiation, transdifferentiation and tissue regeneration. MERLoT simplifies and enhances downstream analysis in multiple ways. By utilizing an explicit tree structure, selecting subgroups of cells that belong to different tree segments or finding differentially expressed genes becomes straightforward. Apart from deriving the tree, MERLoT also imputes and interpolates gene expression, drastically reducing noise and alleviating the problem of gene dropout and hence enhancing downstream analysis. A recent publication described how diverse connective tissue cell types regenerated axolotl limbs after amputation by converging to the homogeneous transcriptional signatures of multipotent progenitor cells (29). The authors used MERLoT to reconstruct the lineage tree, impute the gene expression values as a function of pseudotime, and study how gene expression levels changed among the different groups of cells in the process. Here, we have derived a GCN for a dataset representing fibroblast to neuron transdifferentiation as well as zygote to blastocyst differentiation (see Supplemental Note 5). We show that using MERLoT's imputed expression values improves capture of gene-gene correlations, and could be used as input for more sophisticated methods that aim to reconstruct gene regulatory networks.

While our benchmark showed that MERLoT's default approach leads to satisfactory results for a wide variety of topologies and expression matrices of various sizes, we are aware that when dealing with real data most methods don't work out of the box. Currently, researchers rely on external, expert knowledge about the studied systems and manually optimize strategies on every step of the process (gene selection filtering, cell quality filtering, different manifold embeddings, and tuning of parameters related to all of these). MERLoT is flexible enough to allow supervision at different parts of the analysis pipeline, while providing default strategies that are robust enough to be used for exploratory analysis.

Single-cell RNA sequencing enables the study of time-dependent processes in unprecedented detail. With the help of tools like MERLoT, we can overcome the high noise and non-linearities in the data, reconstruct the cellular lineage trees, and follow the change of gene expression over developmental time. Such time course gene expression profiles pave the way for the reconstruction of gene regulatory networks and eventually their quantitative modelling, which will profoundly advance our understanding of developmental processes.

Conflict of interest statement. None declared.

Data availability The 10 simulation sets with 100 simulated differentiations each are available at `http://wwwuser.gwdg.de/~compbiol/merlot/`. The code necessary to run the benchmark on the simulations as well as instructions about how to set up a similar benchmark are available at `https://github.com/soedinglab/merlot-scripts`. Formatted expression data for the three datasets in Fig. 2 are available at: `https://github.com/soedinglab/merlot/tree/master/inst/example/`

REFERENCES

1. Klein, A. M., Mazutis, L., Akartuna, I., Tallapragada, N., Veres, A., Li, V., Peshkin, L., Weitz, D. A., and Kirschner, M. W. (2015) Droplet barcoding for single-cell transcriptomics applied to embryonic stem cells. *Cell,* **161**(5), 1187–1201.
2. Macosko, E. Z., Basu, A., Satija, R., Nemesh, J., Shekhar, K., Goldman, M., Tirosh, I., Bialas, A. R., Kamitaki, N., Martersteck, E. M., et al. (2015) Highly parallel genome-wide expression profiling of individual cells using nanoliter droplets. *Cell,* **161**(5), 1202–1214.
3. Trapnell, C. (2015) Defining cell types and states with single-cell genomics. *Genome research,* **25**(10), 1491–1498.
4. Cao, J., Packer, J. S., Ramani, V., Cusanovich, D. A., Huynh, C., Daza, R., Qiu, X., Lee, C., Furlan, S. N., Steemers, F. J., et al. (2017) Comprehensive single-cell transcriptional profiling of a multicellular organism. *Science,* **357**(6352), 661–667.
5. Regev, A., Teichmann, S., Lander, E. S., Amit, I., Benoist, C., Birney, E., Bodenmiller, B., Campbell, P., Carninci, P., Clatworthy, M., Clevers, H., Deplancke, B., Dunham, I., Eberwine, J., Eils, R., Enard, W., Farmer, A., Fugger, L., Gottgens, B., Hacohen, N., Haniffa, M., Hemberg, M., Kim, S. K., Klenerman, P., Kriegstein, A., Lein, E., Linnarsson, S., Lundeberg, J., Majumder, P., Marioni, J., Merad, M., Mhlanga, M., Nawijn, M., Netea, M., Nolan, G., Pe'er, D., Philipakis, A., Ponting, C. P., Quake, S. R., Reik, W., Rozenblatt-Rosen, O., Sanes, J. R., Satija, R., Shumacher, T., Shalek, A. K., Shapiro, E., Sharma, P., Shin, J., Stegle, O., Stratton, M., Stubbington, M. J. T., van Oudenaarden, A., Wagner, A., Watt, F. M., Weissman, J. S., Wold, B., Xavier, R. J., and Yosef, N. (2017) The Human Cell Atlas. *bioRxiv,*.
6. Wagner, A., Regev, A., and Yosef, N. (2016) Revealing the vectors of cellular identity with single-cell genomics. *Nature biotechnology,* **34**(11), 1145–1160.
7. Rostom, R., Svensson, V., Teichmann, S. A., and Kar, G. (2017) Computational approaches for interpreting scRNA-seq data. *FEBS Letters,*.
8. Saelens, W., Cannoodt, R., Todorov, H., and Saeys, Y. (2018) A comparison of single-cell trajectory inference methods: towards more accurate and robust tools. *bioRxiv,* p. 276907.
9. Coifman, R. R., Lafon, S., Lee, A. B., Maggioni, M., Nadler, B., Warner, F., and Zucker, S. W. (2005) Geometric diffusions as a tool for harmonic analysis and structure definition of data: Diffusion maps. *Proceedings of the National Academy of Sciences of the United States of America,* **102**(21), 7426–7431.
10. Angerer, P., Haghverdi, L., Büttner, M., Theis, F. J., Marr, C., and Buettner, F. (2016) destiny: diffusion maps for large-scale single-cell data in R. *Bioinformatics,* **32**(8), 1241–1243.
11. Papadopoulos, N., Parra, R. G., and Soeding, J. (2019) PROSSTT: Probabilistic simulation of single-cell RNA-seq data for complex differentiation processes.. *Bioinformatics,*.
12. Zappia, L., Phipson, B., and Oshlack, A. (2017) Splatter: simulation of single-cell RNA sequencing data. *Genome biology,* **18**(1), 174.
13. Cayley, A. (1857) XXVIII. On the theory of the analytical forms called trees. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science,* **13**(85), 172–176.
14. Saitou, N. and Nei, M. (1987) The neighbor-joining method: a new method for reconstructing phylogenetic trees.. *Molecular biology and evolution,* **4**(4), 406–425.
15. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011) Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research,* **12**, 2825–2830.
16. Gorban, A. and Zinovyev, A. (2005) Elastic principal graphs and manifolds and their practical applications. *Computing,* **75**(4), 359–379.
17. Gorban, A. N., Sumner, N. R., and Zinovyev, A. Y. (2007) Topological grammars for data approximation. *Applied Mathematics Letters,* **20**(4), 382–386.
18. Trapnell, C., Cacchiarelli, D., Grimsby, J., Pokharel, P., Li, S., Morse, M., Lennon, N. J., Livak, K. J., Mikkelsen, T. S., and Rinn, J. L. (2014) Pseudo-temporal ordering of individual cells reveals dynamics and regulators of cell fate decisions. *Nature biotechnology,* **32**(4), 381.
19. Paul, F., Arkin, Y., Giladi, A., Jaitin, D. A., Kenigsberg, E., Keren-Shaul, H., Winter, D., Lara-Astiaso, D., Gury, M., Weiner, A., et al. (2015) Transcriptional heterogeneity and lineage commitment in myeloid progenitors. *Cell,* **163**(7), 1663–1677.
20. Guo, G., Huss, M., Tong, G. Q., Wang, C., Sun, L. L., Clarke, N. D., and Robson, P. (2010) Resolution of cell fate decisions revealed by single-cell gene expression analysis from zygote to blastocyst. *Developmental cell,* **18**(4), 675–685.
21. Velten, L., Haas, S. F., Raffel, S., Blaszkiewicz, S., Islam, S., Hennig, B. P., Hirche, C., Lutz, C., Buss, E. C., Nowak, D., et al. (2017) Human haematopoietic stem cell lineage commitment is a continuous process. *Nature cell biology,* **19**(4), 271.
22. Treutlein, B., Lee, Q. Y., Camp, J. G., Mall, M., Koh, W., Shariati, S. A. M., Sim, S., Neff, N. F., Skotheim, J. M., Wernig, M., et al. (2016) Dissecting direct reprogramming from fibroblast to neuron using single-cell RNA-seq. *Nature,*.
23. Pons, P. and Latapy, M. (2005) Computing communities in large networks using random walks. In *International symposium on computer and information sciences* Springer pp. 284–293.
24. Dijkstra, E. W. (1959) A note on two problems in connexion with graphs. *Numerische mathematik,* **1**(1), 269–271.
25. Qiu, X., Mao, Q., Tang, Y., Wang, L., Chawla, R., Pliner, H. A., and Trapnell, C. (2017) Reversed graph embedding resolves complex single-cell trajectories. *Nature methods,* **14**(10), 979–982.
26. Welch, J. D., Hartemink, A. J., and Prins, J. F. (2016) SLICER: inferring branched, nonlinear cellular trajectories from single cell RNA-seq data. *Genome biology,* **17**(1), 106.
27. Ji, Z. and Ji, H. (2016) TSCAN: Pseudo-time reconstruction and evaluation in single-cell RNA-seq analysis. *Nucleic acids research,* **44**(13), e117–e117.
28. Street, K., Risso, D., Fletcher, R. B., Das, D., Ngai, J., Yosef, N., Purdom, E., and Dudoit, S. (Jun, 2018) Slingshot: cell lineage and pseudotime inference for single-cell transcriptomics. *BMC Genomics,* **19**(1), 477.
29. Gerber, T., Murawala, P., Knapp, D., Masselink, W., Schuez, M., Hermann, S., Gac-Santel, M., Nowoshilow, S., Kageyama, J., Khattak, S., et al. (2018) Single-cell analysis uncovers convergence of cell identities during axolotl limb regeneration. *Science,* p. eaaq0681.

Supplementary Information
Reconstructing complex lineage trees from scRNA-seq data using MERLoT

R. Gonzalo Parra[1,2,†,*], Nikolaos Papadopoulos[1,†], Laura Ahumada-Arranz[1], Jakob El Kholtei[1], Noah Mottelson[1], Yehor Horokhovsky[1], Barbara Treutlein[3], and Johannes Soeding[1,*]

[*]corresponding author
[1]†equal contribution
[1]Quantitative and Computational Biology Group, Max Planck Institute for Biophysical Chemistry, Am Fassberg 11, 37077, Goettingen, Germany
[2]Genome Biology Unit, European Molecular Biology Laboratory, Meyerhofstraße 1, 69117, Heidelberg, Germany
[3]Department of Evolutionary Genetics, Max Planck Institute for Evolutionary Anthropology, Deutscher Platz 6, 04103, Leipzig, Germany

Methods

Suppl. Note 1: Lineage Tree Reconstruction by MERLoT

Given an expression matrix with N cells as rows and G genes as columns, a manifold embedding can be performed using techniques like Diffusion Maps or DDRTree embedding. Subsequently, informative dimensions can be kept in order to reduce dimensionality. Once cells are embedded into the low dimensional space, MERLoT can perform a lineage tree topology reconstruction following three steps which will be detailed in the following sections: (1) Calculating a Scaffold Tree with the location of endpoints, branchpoints and their connectivity. (2) Smoothing of the Scaffold Tree by using an EPT in the low dimensional manifold. (3) Embedding the EPT into the high-dimensional gene expression space and assigning pseudotime values to the cells.

Tree topology, is the set of endpoints and branchpoints that constitute the tree structure of the tree, i.e, a linear trajectory is only composed of a single branch with two endpoints and no branchpoints. A single bifurcated topology contains 3 branches that are connected via a branchpoint.

branches: set of connected nodes (cells) that constitute a branch that is defined as a linear segment of the tree contained between two endpoints or branchpoints. endpoint: is the node on which a branch ends. branchpoint: it is a node that connects three or more branches (in case of binary bifurcations or higher order multifurcations).

Scaffold Tree Reconstruction

Matrix of shortest paths between cells: We used Dijkstra's shortest path algorithm as implemented in the `csgraph` module from the `scipy` python library to calculate the shortest paths between all pairs of cells i and j based on their squared Euclidean distance d_{ij}^2. We use d_{ij}^2 instead of d_{ij} given that in a fully connected graph the shortest path between two nodes, based on d_{ij}, corresponds to the direct path between them. The `csgraph shortest_path` function returns the length D_{ij} of the shortest path that connects cells i and j. We extended the function to also return the number of cells S_{ij} on the shortest path connecting cells i and j. The modified code for `csgraph` is available at `github.com/soedinglab/csgraph_mod`.

Dijkstra's shortest-path-first algorithm:
Given a graph G where the distance between two nodes u, v is given by $dist(u, v)$, the algorithm will find the shortest paths from an initial origin node to all other nodes n in the graph. The algorithm initializes all nodes (except the origin) as unvisited; their distance $D[n]$ from the origin is infinite, and the distance of the origin to itself is zero. The origin is the current node c.

Consider all of the neighbours v of c, and calculate their distances from *origin* through the current node. If $D[c] + d(c, v) < D[v]$, then update $D[v]$ to $D[c] + d(c, v)$ and add c as the parent node of v. Repeat this until all nodes are visited. Afterwards, the shortest path that leads from node n to the origin is found by following the parent nodes of n until the origin is reached.

Endpoints search: MERLoT does not require users to define a starting point in order to reconstruct the tree topology. The first two endpoints in the tree correspond to the pair of cells k, l that maximize the number of cells S_{kl} on the shortest path between them (Fig. S1A): $(k, l) = \arg\max\{S_{kl} \,:\, 1 \leq k \leq l \leq N\}$. In case of ties, the pair of cells with the longest shortest path distance will be selected.

The next endpoint to be selected will correspond to the cell n that maximizes $s_{\mathcal{E}}(n)$, the number of cells being added to the scaffold tree structure. To compute $s_{\mathcal{E}}(n)$ we note that the new endpoint n must branch off from an internal node (cell) of the tree whose next-neighbour nodes must lie on the path of two already selected endpoints k and l. Therefore, the increase in number of cells is 0.5 times the minimum of the cells between k and l *via* n minus the cells between k and l, minimized over all pairs of endpoints (k, l) in the set of already selected endpoints $\mathcal{E}$.

$$s_{\mathcal{E}}(n) := 0.5 \times \min\{S_{kn} + S_{nl} - S_{kl} \,:\, k, l \in \mathcal{E}\}. \tag{1}$$

Stop criterion for endpoint search: In auto mode every time a new endpoint is selected we evaluate if $\max\{s_{\mathcal{E}}(n',) : 1 \leq n' \leq N, n' \notin \mathcal{E}\} > \sqrt{N}$ holds true. Otherwise, we calculate the branchpoints and tree connectivity for the endpoints in $\mathcal{E}$, including n, using the methodology explained in the next subsection. After this, all cells are mapped to their closest branch. If the branch added by the selected n endpoint contains more than `MinBranchCells`$= \sqrt{N}$ cells mapped to it, the branch is kept in the tree scaffold structure and the endpoints search is repeated. Otherwise, the endpoint search terminates and n is discarded as endpoint. The `MinBranchCells` threshold can be modified by the user. Alternatively, instead of using a stop criterion, users can set the number of endpoints that are aimed to be found (fixed mode) regardless of the branch lengths.

Branchpoints search and tree connectivity definition

Once endpoints are found, we apply the in combination with a heuristic criterion to find the cells that best represent the branchpoints in the tree structure. By doing this, we also detect the tree connectivity among endpoints and branchpoints

Given a set of endpoint and branchpoint cells we iterate the following steps: (1) Use the Neighbour Joining (NJ) criterion [1] to select the pair of (k, l) cells that will be joined *via* a branchpoint next. 2) Find the cell m that best represents the branchpoint between k and l and add the $l - m$ and $m - k$ edges to the tree (Fig. S2).

Let $\mathcal{V}$ be the set of yet unprocessed endpoint and branchpoint nodes of the tree. We initialize $\mathcal{V} \leftarrow \mathcal{E}$ with the endpoint set $\mathcal{E}$ determined in the previous subsection.

(1) Given the matrix D_{kl} of shortest path lengths between nodes in $\mathcal{V}$ (section 1.1), the NJ criterion allows us to pick two nodes k, l in $\mathcal{V}$ that are guaranteed to be next neighbours and therefore can be linked via a single branchpoint. The nodes to be joined are chosen such that $(k, l) = \arg\min d^{\mathrm{NJ}}_{k,l}$ where d^{NJ}_{kl} is the neighbour-joining distance,

$$d^{\mathrm{NJ}}_{kl} := D_{kl} - \frac{1}{|\mathcal{V} - 2|} \sum_{m \in \mathcal{V}} (D_{mk} + D_{ml})\,. \tag{2}$$

(2) We determine the branchpoint cell m as the one that minimizes the sum of distances to nodes k, l and the mean distances to all other nodes included in $\mathcal{V}$,

$$m = \arg_m \min \left\{ D_{km} + D_{lm} + \frac{1}{|\mathcal{V} - 2|} \sum_{n \in \mathcal{V} \setminus \{k,l\}} D_{nm} \right\}. \tag{3}$$

Next, k and l are removed from $\mathcal{V}$ and the new branchpoint m is added to $\mathcal{V}$, $\mathcal{V} \leftarrow \mathcal{V} \setminus \{k, l\} \cup \{m\}$.. Also, the edges $l - m$ and $m - k$ are added to the tree (Fig. S2).

We iterate (1) and (2) until $|\mathcal{V}| = 2$, which means no further branchpoints exist. After termination, we have determined the tree topology with its $|\mathcal{E}|$ endpoints and $|\mathcal{E}| - 2$ branchpoints, each represented by a cell. The same cell can be detected more than once as a branchpoint and hence bifurcations with higher orders than binary ones are possible.

Suppl. Note 2: Scaffold Tree pseudocode

```
INPUT
CellCoordinates // N: number of cells, G: number of components

// 1. Calculate cell pairwise squared euclidean distances.
$D_e^2$ = CalcEuclideanDistances(CellCoordinates)^2

// 2. For every pair of cells c_i, c_j calculate:
```

```
// Shortest path euclidean distance D_p(c_i, c_j)
// Shortest geodesic distance, D_g(c_i, c_j)
(D_p, D_g) = DijkstraShortestPath(D_e)

// 3. Calculate endpoints E
// a) First pair of endpoints: Find pair of cells k, l with maximum
// geodesic distance D_g
(k,l) = argmax(D_g)
E = list()
E.append(k), E.append(l)

// Find further endpoints
// Stop criteria depends on the MERLoT flavor - either stop after a certain number
// of branches has been reached (fixed mode) or after new branches add less cells
// than a cutoff.
while stop_criteria == False:
  // initialize vector containing endpoint score
  s_E = zeros(N)
  for n in N:
    // calculate how many cells would be added to the tree if
    // n was a new endpoint, in terms of shortest paths
    aux = zeros(length(E), length(E))
    for e_1 in E:
      for e_2 in E:
        aux[e_1, e_2] = D_g[e_1, n] + D_g[n, e_2] - D_g[e_1, e_2]

    s_E[n] = 0.5 * min(aux)
    E.append(max(s_E))

// 4. Find branchpoints and define topology edges in list B
// Initialize V which will contain the remaining endpoints to be joined during each step
V = E
B = list()
// List of edges. We need this for the connectivity.
Edges=list()
while length(V) >= 2:
  // find pair of endpoints to be joined
  for e_1 in 1:length(V):
    for e_2 in 1:length(V):
      // calculate Neighbour Joining distance between endpoints e_1 and e_2 according
      // to equation (2) in Materials and Methods
      $D_NJ[e_1, e_2] = calc_NJ_distance(V[e_1], V[e_2])

  // endpoints to be joined
  (e_1, e_2) = min(D_NJ)
  // select cell m that minimizes D_NJ as branchpoint between endpoints
  // e_1 and e_2 according to equation (3) in materials and methods
  b = argmin(D_NJ[e_1, e_2])

  B.append(b)

  // Add edges to tree topology
  Edges.append([e_1, b])
  Edges.append([e_2, b])

  // replace joined endpoints from V with b
  V.remove(e_1)
  V.remove(e_2)
  V.append(b)
```

```
// no further branchpoints to be searched, append the last edge to the tree topology.
Edges.append (V[1], V[2])
return(E, B, Edges)
```

Elastic Principal Tree in the Low Dimensional Manifold

We use the coordinates of the endpoints and branchpoints of the scaffold tree and their connectivity to initialize the EPT. Further nodes are then added by the EPT algorithm by iterative bisection of edges until the specified number of k support nodes is reached. This procedure cannot modify the number of endpoints and branchpoints specified at the initialization step.

In every iteration an energy potential defined as follows is minimized:

$$U = \mathrm{MSE} + \mu U_E + \lambda U_R. \tag{4}$$

MSE (mean squared error) is the sum of squared distances of cells to their closest tree support nodes. U_E and U_R are two regularization terms that ensure that we learn trees with regularly spaced points and with edges without kinks, respectively.

We performed a grid search around the EPT default values in order to optimize μ and λ and visually examined the reconstructed EPTs on the datasets shown in Fig. 2. Using $k = 100$, we obtained $\mu_0 = 0.0025$ and $\lambda_0 = 0.8 \cdot 10^{-9}$. If the number of nodes used for calculating the EPT is changed, μ and λ are adjusted according to $\mu = (k-1)\mu_0$ and $\lambda = (k-2)^3\lambda_0$. All reconstructions in our benchmark were performed with the standard function using $k = 100$. However, for some particular topologies μ and λ might need to be tuned in order to produce optimal results, in particular if k is increased a lot. Alternatively, MERLoT can bisect the edges in a given EPT, by additional nodes producing a new EPT with almost $2k$ support nodes. Note that these are special cases.

Scaling of μ and λ with the number of tree points K. We would like to be able to change K without changing the shape of the tree. Say we change from K tree points to $2K-1$. Then we want the even-numbered points $y'_0, y'_2, y'_4, \ldots, y'_{2K-1}$ to come to lie exactly where previously we had points $y_0, y_1, \ldots, y_{K-1}$. Since the data term will stay very nearly the same if the points were already close to each other, the bending and stretching energies should also stay the same. How do we need to scale $\mu(K)$ and $\lambda(K)$ in order to keep these terms the same?

Let us start with the stretching energy. The distances between neighbouring points will be half the previous distances and their squares will be a quarter. On the other hand, we will have 2K-2 such terms instead of $K-1$, so twice more. Therefore, if we scale

$$\mu(K) = (K-1)\mu_0\,, \tag{5}$$

the total stretching energy will be conserved.

Regarding the bending energy, the terms $||y_k - (y_{\mathrm{pre}(k)} + y_{\mathrm{suc}(k)})/2||$ can be approximated as $r_k(1-\cos(\theta_k/2))$, where r_k is the bending radius at point y_k and θ is the angle of direction change between y_{k-1} and y_{k+1} (which is equal to the angle between $\overline{Cy_{k-1}}$ and $\overline{Cy_{k+1}}$, where C is the center of the circle of the bending radius through y_{k-1}, y_k, and y_{k+1}). Since θ is halved upon going from K to $2K-1$ tree points, the distance $r_k(1-\cos(\theta_k/2)) \approx r_k\theta_k^2/8$ will be 4 times smaller and the squared distance will get 16 times smaller. Since there will be $2K-3$ instead of $K-2$ such squared terms, we need to scale

$$\lambda \approx (K-2)^3\lambda_0 \tag{6}$$

in order to keep the bending energy approximately constant.

Elasticity hyperparameters: We selected appropriate default elasticity hyperparameters for the elastic and embedded tree by performing a grid search around the default EPT hyperparameter values. If N_y is the number of nodes of the elastic tree, we define $\mu_{\mathrm{el}} = N_y\mu_0$ and $\lambda_{\mathrm{el}} = ((N_y-2)^3)\lambda_0$, with $\mu_0 = 0.0025$ and $\lambda_0 = 0.8 \cdot 10^{-09}$. For the embedded tree, we use $\mu_{\mathrm{emb}} = (N_y-1)\cdot\mu_0\cdot\phi_\mu$ and $\lambda_{\mathrm{emb}} = (N_y-2)^3\cdot\lambda_0\cdot\phi_\lambda$, where $\mu_0 = 0.00625$, $\lambda_0 = 2.03 \cdot 10^{-09}$, and $\phi_\mu = \phi_\lambda = 20$. In the benchmark, where N_y was 100, this amounted to values of $\mu_{\mathrm{el}} = 0.0025$, $\lambda_{\mathrm{el}} \approx 0.00075$, $\mu_{\mathrm{emb}} = 12.375$, and $\lambda_{\mathrm{emb}} \approx 0.038$. The absence of wiggles in 2D or 3D projections of the tree and gene expression profile plots strongly indicate that the trees did not overfit the cell density.

In the "deep" benchmark, beyond the default values we also tested MERLoT with the default ElPiGraph.R hyperparameter values for the elastic tree functions, $\mu_{\mathrm{ElPi}} = 0.1$ and $\lambda_{\mathrm{ElPi}} = 0.01$. Performance in pseudotime prediction is indistinguishable for both approaches (Fig. S7, right panel). In branch assignment, while the elastic trees (red, cyan) perform at similar levels, the ElPiGraph.R embedded trees tend to have higher average scores for higher order topologies.

After the benchmark was finished we noticed that when using the ElPiGraph.R parameters MERLoT produced embedded trees with many wiggles, indicative of overfitting. We explore this in more detail in two R notebooks (`http://wwwuser.gwdg.de/~compbiol/merlot/examples/`), using data from [2] and [3], respectively.

Suppl. Note 3: Benchmark on Synthetic Datasets

Suppl. Note 3:.1 Simulating count data of branching processes with PROSSTT

PROSSTT generates a simulated scRNA-seq dataset in four steps:

1. Generate tree: We sample the number of genes from a discrete uniform distribution between 100 and 1000. These are typical numbers left in real datasets after filtering out uninformative genes.

Each tree segment in every simulation had a pseudotime length of 50, corresponding to 50 cells on the branch (in homogeneous sampling model). This length allows the expression programs to diffuse enough to be distinct from each other. Starting at 2 bifurcations, alternative segment connectivity possibilities become available (Fig. S8)). We chose the lineage tree topology randomly for every simulation.

2. Simulate average gene expression along tree: PROSSTT models relative gene expression as a linear mixture of a small number of expression programs. For each tree segment, we simulate the time evolution of expression programs as a random walk with momentum term. Each simulation uses $K = 5b + u$ expression programs, where b is the number of branchpoints and u is drawn from a uniform integer distribution $\mathcal{U}\{3, 20\}$. Each program contributes to the expression of every gene, with weights drawn from a gamma distribution (shape parameter a scales inversely with number of programs, rate parameter b is 1). A scaling factor (library size) was sampled for each cell and multiplied to the average gene expression values. We used a log-normal distribution with $\mu = 0$ and $\sigma = 0.7$.

3. Sample cells from tree: PROSSTT can sample cells in the tree according to a given density function. Here we used a uniform density and drew $50 \times b$ cells, where b is the number of branches.

4. Simulate UMI counts: We simulate UMI counts using a negative binomial distribution. Following [4] and [5], we make the variance σ_g^2 depend on the expected expression μ_g as $\sigma_g^2 = \alpha_g \mu_g^2 + \beta_g \mu_g$. The α_g and β_g values were sampled from log-normal distributions with $\mu_{a_g} = 0.2, \sigma_{a_g}^2 = 1.5$ and $\mu_{b_g} = 2, \sigma_{b_g}^2 = 1.5$ respectively. These values are typical for single-cell RNA sequencing UMI counts. For more information about default parameters choices and the algorithm, please refer to [6].

We provide the scripts used to create the simulations (`https://github.com/soedinglab/merlot-scripts`) as well as the simulations themselves (http://wwwuser.gwdg.de/∼compbiol/merlot/).

Suppl. Note 3:.2 Simulating count data of branching processes with Splatter

We chose Splatter simulation hyperparameters to mirror those in the PROSSTT simulations as closely as possible.

- **Global parameters:** the number of genes was picked from the same discrete uniform distribution $\mathcal{U}\{100, 1000\}$. A random seed was set and included to ensure reproducibility.
- **Batch parameters:** The number of batch cells (substitutes total number of cells if only one batch is present, as is the case here) was set to $100 \times b$, where b is the number of branches. This is twice the number sampled for PROSSTT simulations, which increased the robustness of the dimensionality reduction. The other batch parameters were left to their default values.
- **Group parameters:** The number of groups was set to the number of branches, and the occurrence probability of each group was set to $1/b$.
- **Differential expression parameters:** Since we were only interested in simulating informative genes, we set the probability of differential expression per gene to 1. All other parameters were left at their default values.
- **Differentiation path parameters:** The topology of the lineage tree was input as a vector of originating points per branch (`path.from` parameter). Branch lengths were set to 50 via `path.length`.

Suppl. Note 3:.3 Assessing Methods Performance

Tree reconstruction tools need to succeed at two intertwined tasks: to arrange all cells according to their internal developmental time, while also detecting and separating different branches of the tree. Ideally, one would evaluate algorithm performance on both tasks at once. However, we were not able to find such a measure and so evaluate branch assignment and pseudotime prediction separately.

Branch assignment We treated branch assignment evaluation as a clustering problem. We can consider all cells mapped to a given branch of a tree as a cluster and compare the set of labels produced by PROSSTT with those that

are predicted by each algorithm. Given a cell c, let $Tr(c)$ be the cluster identity of c in the ground truth and $A(c)$ the cluster assigned to c by the algorithm. We define a pair of two cells c_i, c_j as a...

$$\begin{array}{ll} \text{true positive (TP)} & \text{if} \quad \mathrm{Tr}(c_i) = \mathrm{Tr}(c_j) \wedge \mathrm{A}(c_i) = \mathrm{A}(c_j) \\ \text{true negative (TN)} & \text{if} \quad \mathrm{Tr}(c_i) \neq \mathrm{Tr}(c_j) \wedge \mathrm{A}(c_i) \neq \mathrm{A}(c_j) \\ \text{false positive (FP)} & \text{if} \quad \mathrm{Tr}(c_i) \neq \mathrm{Tr}(c_j) \wedge \mathrm{A}(c_i) = \mathrm{A}(c_j) \\ \text{false negative (FN)} & \text{if} \quad \mathrm{Tr}(c_i) = \mathrm{Tr}(c_j) \wedge \mathrm{A}(c_i) \neq \mathrm{A}(c_j) \end{array}$$

Using these four values, many popular clustering indices can be computed (Fig S4): the F1 measure (Fig. S4A) is $2PR/(P+R)$, (where P is the precision $TP/(TP+FP)$ and R is the recall $TP/(TP+FN)$), the MCC (Fig. S4B) is

$$\frac{(TP \cdot TN - FP \cdot FN)}{\sqrt{((TP+FP)(TP+FN)(TN+FP)(TN+FN))}},$$

and the Jaccard Index (Fig. S4C) is $TP/(TP+FP+FN)$

While all the aforementioned indices and measures are well established, they are suboptimal performance indicators for the problem at hand, since they don't take cluster structure into consideration. As the simulated lineage trees become bigger and more complex, the number of cell pairs that are not in the same tree segment is going to grow much faster than the number of cell pairs in different tree segments. The number of TPs will grow much slower than the number of TNs, something that will, for example, inflate the the MCC index. Additionally, the number of possible FNs becomes much higher with each additional segment added to the tree, something that affects the Jaccard index and the F1 score. In short, these measures, while they produce consistent results, they don't take into account the number of clusters in the data, and as such are suboptimal descriptors of clustering performance. In our opinion the NMI is best for assessment of branch assignment, since it captures the amount of information present in the original clustering that was recovered by the prediction (values between 0 and 1). It corrects the effect of agreement between clusters that is due to chance by using a hypergeometric background distribution and punishes overbranching and merging branches almost equally [7]. Given the predicted and real cluster assignments U and V, NMI is defined as

$$\mathrm{NMI}(U,V) = \frac{\mathrm{MI}(U,V) - \mathbb{E}[\mathrm{MI}(U,V)]}{\max\{H(U),H(V)\} - \mathbb{E}[\mathrm{MI}(U,V)]}.$$

where $H(U)$ is the entropy of U, MI(U, V) is the mutual information of U, V and $\mathbb{E}$ denotes the expectation value under the null model of independent U and V.

Pseudotime prediction Evaluating the performance of the different methods consists of quantifying the degree of agreement between the true/labeled and the predicted orderings provided by the different algorithms. Pseudotime only establishes a partial ordering on cells and no absolute time. Therefore, cells on branches not passed through one after the other cannot be compared. We find the longest path in the tree (from the root to a leaf) and compare the predicted pseudotime with the simulated one for the cells on this path.

In this sense, pseudotime prediction assessment is a comparison of ordered sequences, and we follow the suggestions of [8] by using the Goodman-Kruskal index and the Kendall index (weighted and unweighted). All four indices count how many pairs of cells have been ordered correctly (S_+) or incorrectly (S_-) and produce similar results for the benchmark. The unweighted Goodman-Kruskal index is the simplest approach: $(S_+ - S_-)/(S_+ + S_-)$.

Complexity of predicted topologies We analyzed the predicted topology complexity, measured in number of branchpoints per lineage tree (Fig. S9). TSCAN and SLICER usually predict more branches than the ones being simulated. Monocle2, Slingshot, and MERLoT mostly overbranch for the simpler topologies but reverse the trend towards more complicated ones. The only notable difference between the "lean" and Splatter sets is that Slingshot_DDRTree detects more branches in "lean".

Suppl. Note 3:.4 Benchmarked Methods and Parameters

Monocle2: Monocle2 (version 2.10.1) uses a reverse graph embedding (RGE) technique called DDRTree [9] to create a lineage tree and map points from the low-dimensional embedding of the tree to the original gene space. Based on Monocle's vignette, we used the `negbinomial()` expression family for the data and the proposed defaults for lower detection limit (1), minimum expression (`detectGenes` function, 0.1), mean expression threshold (gene ordering, 0.5), and empirical dispersion threshold ($2 \cdot dispersion\_fit$).

Generally, a successful dimensionality reduction will capture a topology with B bifurcations in its first $d = B + 1$ dimensions. While Monocle2 runs DDRTree by default on two dimensions, in our benchmarks we used $B + 1$ dimensions, as it performed better. DDRTree did not always return d dimensions; if fewer dimensions were provided we used the maximum possible number.

Monocle2 assigns a branch identity to each cell in the `State` column of the phenotypic data table (`pData`). It calculates pseudotime as distance from one of the endpoints of the lineage tree it produces. In order to pick the correct endpoint,

we checked if the cell with simulated pseudotime $t_0 = 0$ was in an outer branch; if true, the corresponding endpoint was assigned t_0. In the rare event that it was placed in an inner branch, we used Monocle's chosen starting `State` for pseudotime calculation.

SLICER: SLICER uses Locally Linear Embedding (LLE) to perform dimensionality reduction and uses a neighbour graph to order cells according to their distance from a user-specified starting cell (pseudotime prediction). It uses geodesic entropy to recursively detect branches.

We used SLICER version 0.2.0 (commit `cb1be8a`) by following the instructions that accompany the software on its github page (`https://github.com/jw156605/SLICER/`). We used the software's gene selection process as-is and used the selected genes to determine the best k value for the LLE, with a $k_{\min}$ value of 5. Much like with Monocle2, using $d = B+1$ LLE dimensions (over the default 2) for a dataset with B bifurcations improved performance. We used the same k value for the creation of the low-dimensional k-nearest neighbour graph as we did for LLE. For every simulation, we used the cell with labeled pseudotime $t_0 = 0$ as the starting point of the `cell_order` function, which predicts pseudotime for each cell. Finally, we used the same start point for the branch assignment step (`assign_branches`). This step very often failed to execute; these cases were assigned the worst possible score of each branch assignment measure. The issue was reported to the authors (`https://github.com/jw156605/SLICER/issues/7`) but until the time of this writing was not resolved.

Destiny: Destiny produces a dimensionality reduction and a pseudotime prediction based on it. We used the destiny diffusion map space as input for MERLoT, and benchmarked destiny's Diffusion Pseudotime (DPT). We used destiny (version 2.6.1) as described in the Bioconductor vignette. First we normalized the input data by correcting for library size and then log-transformed them. The only free parameter is the number k of nearest neighbours. By default, destiny uses a heuristic to determine it. For a dataset with B bifurcations we gave destiny $d = B + 1$ dimensions of the diffusion map to determine pseudotime, following the same reasoning as with Monocle2 and SLICER. We retrieved pseudotime predictions from destiny by calling the `DPT` (Diffusion Pseudo-Time) function on the diffusion map object and then using the dimension which corresponded to diffusion pseudotime distances from the cell with pseudotime $t_0 = 0$.

MERLoT: We ran different flavors of MERLoT on embeddings from different dimensionality reduction tools:

1. **MERLoT + diffusion maps:** destiny was run to produce diffusion maps for MERLoT. The same protocol as in section Suppl. Note 3:.4 was used, except for the selection of free parameter k, which was done by a simple optimization. For a simulated lineage tree with B bifurcations we tested values of k between 5 and 100 and kept the k value that maximized the drop-off after the $d + 1$'th eigenvalue of the diffusion map, where $d = B + 1$.
2. **MERLoT + DDRTree:** Monocle2 was run in order to produce DDRTree coordinates. The number of coordinates to be used to reconstruct the lineage tree was selected as described in section Suppl. Note 3:.4.

Each of these two options was combined with two different ways in which MERLoT finds the number of endpoints in the dataset.

1. **MERLoT auto:** MERLoT is run without the specification of the number of endpoints in the tree. MERLoT will use its internal branch length heuristic to determine new branches. The algorithm stops finding new endpoints when the new branch aimed to be included in the tree structure does not contain more than $\sqrt{(N)}$ number of cells, with N being the total number of cells in the dataset.
2. **MERLoT fixed:** MERLoT is run with a specified number of endpoints (tree leaves) to be found. MERLoT will ignore its internal branch length heuristic and will keep searching for branches until it reaches the specified number of endpoints regardless of how many cells are mapped to them.

All 2×2 combinations were tested (MERLoT on diffusion maps auto/fixed and MERLoT on DDRTree coordinates auto/fixed). The benchmark was performed with commit `9a9fc93` of the "scaffold" branch.

TSCAN: TSCAN (version 1.16.0) is a tool that groups similar cells into clusters and then creates an undirected minimum spanning tree (MST) that connects them, a different approach from the other tools assessed in this benchmark. We followed the typical TSCAN pipeline as presented in the tool's vignette on the Bioconductor site. This consists of a preprocessing step (`preprocess` function) followed by clustering and construction of the Minimal Spanning Tree (MST), which happen in the same step (`exprmclust` function). The `exprmclust` function accepts a number of clusters as parameter. If the user inputs a range of possible clusters, TSCAN will pick the number of clusters using the Bayesian Information Criterion.

TSCAN's cluster approach is an issue when comparing TSCAN predictions to simulations that are made using paths (such as PROSSTT or Splatter), since there is no 1-to-1 relationship between branches and the TSCAN clusters. Additionally, when run with default parameters, TSCAN very often predicts ambiguous topologies, where, because the MST is undirected, it is not possible to tell if bifurcations are present or not.

In order to address this concern we ran TSCAN with the maximum number of clusters possible. This was achieved by starting from 50 (a number well above the maximum number of branches in the simulations), and decreasing the number of clusters until a clustering is produced. Afterwards, co-linear clusters are collapsed and the cells of branching clusters are re-assigned to the nearest adjacent clusters.

Unfortunately, this creates other complications, since TSCAN only calculates pseudotime for the longest trajectory (path from endpoint to endpoint) in the tree. After consulting the authors (see `https://github.com/zji90/TSCAN/issues/4`), we ran the clustering step of TSCAN with 2 clusters. This guarantees a linear path that includes all the cells and thus a global pseudotime ordering can be obtained.

Slingshot: Slingshot [10] is a trajectory inference algorithm very similar to TSCAN. In a first stage, it creates a cluster-based MST to identify the key elements of the global structure - the branches and branching points. In a second step, it uses simultaneous principal curves (an extension of the principal curve algorithm proposed by [11]) to construct smooth trajectories that represent paths from the root of the lineage tree root to its leaves. It then assigns pseudotime to cells by projecting them on these principal curves.

Slingshot does not compute a reduced manifold nor does it perform the clustering step. It is left to the user to supply both. For this benchmark, we used the manifold reductions produced by the other tools (diffusion maps by destiny, DDRTree embedding from Monocle 2, PCA from TSCAN). For the clustering, we used `mclust` [12], an R package for model-based clustering. We ran the `mclustBIC` function, allowing the number of clusters to be between 5 and 50 (for the same reasons as TSCAN, see above). This function fits a Gaussian finite mixture model via an expectation-maximization (EM) procedure, and returns an optimal number of cluster means and variances. In the next step, the cells are assigned to the closest cluster centroid according to the EM model that maximizes the Bayes Information Criterion.

Slingshot assigns a pseudotime to each cell in every trajectory it detects. This means that when two trajectories are close to each other, some cells may have multiple pseudotime values. All trajectories start at the same cell though, so in cases where cells have multiple pseudotime values, these are usually very similar and can be averaged without problems, thus creating global pseudotime values for all cells.

Since Slingshot follows the same cluster strategy as TSCAN, it also faces the same problems when it comes to evaluation in the context of PROSSTT. To avoid biasing the benchmark against Slingshot, we followed the same strategy as above, by identifying and collapsing all co-linear clusters and reassigning cells to the branches that are created.

We used Slingshot version 1.0.0, available for Bioconductor 3.8.

During Slingshot runs on the "deep" benchmark we encountered problems with most of the high-order bifurcations. Specifically, Slingshot struggles when determining the cluster-to-cluster distance. The distance measure is $D^T SD$, where D is the difference of the cluster centroids and S is the matrix inverse of the sum of the covariance of each cluster. What happens frequently is that this inversion is impossible due to "computational singularity". For some cases, rerunning the analysis did not produce the same issue. For the rest, we added a small amount of random noise to the diffusion maps. This noise was sampled from $\mathcal{N}(0, d/100)$, where d is the difference between the minimum and maximum absolute value in manifold space. We reported the issue to the developers `https://github.com/kstreet13/slingshot/issues/35`.

Suppl. Note 3:.5 Divergence analysis

While benchmarking method performance on data simulated with Splatter, we noticed that multiple methods did not perform according to expectations. The issue was particularly obvious in the evaluation of pseudotime prediction (Fig. S15D), where Monocle2 sank to the level of random predictions, and MERLoT, which, even though it excelled at branch assignment (Fig. S15B), dropped off to quite low accuracy for large numbers of bifurcations. Additionally, TSCAN, which in the PROSSTT benchmark proved to be competent in branch assignment (Fig. S15A), returned completely nonsensical predictions for data simulated by Splatter.

As these methods have all been applied successfully on real data, we chose to examine the simulations. By visual inspection of the manifold embeddings we observed that the manifold embeddings of Splatter simulations often presented "short-circuits", where parts of the lineage tree seemed to fold back to preceding tree segments, effectively creating cycles. While branches were often separated correctly (i.e. the cell clustering was correct), they were connected in wrong ways, decreasing the pseudotime prediction accuracy.

Since the manifold embeddings are a projection that aims to retain the most important dynamics in the data, we hypothesized that the reason for these short-circuits was that the offending branches did not diverge enough, or even converged towards previous tree segments. In terms of gene expression, this means that differentiating phenotypes (captured transcriptomes) were either not different enough or that they converged towards preceding phenotypes.

To test this hypothesis, we measured the distance of each waypoint (endpoint or branchpoint) from the origin, and normalized it by the average branch length in the path that led to it. The calculations described below were performed on the simulated gene expression data, after normalization for library size and log-transformation (see scripts `divergence_euclidean.R` and `divergence_diffmaps.R`).

As explained in Fig. S16, we retrieved the cells with minimum and maximum pseudotime in each branch (start cells and end cells respectively), calculated their pairwise distances, and defined their average d_b as the length of branch b.

Next, we took the cells with globally minimum pseudotime and calculated their distances from all end cells. This is the minimum direct distance d'_b of each branch b from the origin. We normalized each d'_b with $\bar{d}_b = \frac{1}{|p|}\sum_{b' \in p} d_{b'}$, where p is the path $[b_0, b_1, ...b]$ from the origin branch b_0 to branch b. Effectively, this yields the distance from the origin measured in average branch lengths. Pooling the normalized distances from all paths of equal length (especially since all branches have the same length in the PROSSTT and Splatter simulations) shows how much the change in expression values correlates with pseudotime.

The divergence curve of PROSSTT (Fig. S17A) shows what we expected: monotonic growth (i.e. longer paths are on average further away from the origin) and values above 1, indicating that paths with two branches or more consistently end outside a one-branch-length radius from the origin, something that reduces the possibility of wrong assignments from the methods.

On the contrary, the divergence curve of Splatter (Fig. S17B) stays almost completely below 1 and even shows a slightly negative slope. This means that expression profiles of cells in later differentiation stages don't move further from the origin with increasing pseudotime length. We believe this happens because Splatter does not include co-regulation in its differentiation simulation model. This leads to the differences between branches being completely random, and while this may work to separate two diverging branches from each other, it does not seem to yield realistic diverging cell lineage trees.

As a control, we performed the divergence analysis in the diffusion maps created by destiny for the benchmark (panels G,H in Fig. S17). Diffusion distance was proposed as a measure of cell similarity in the original paper [13], and was the most effective of the embeddings used in this study. We see the same trends as in gene expression space; the divergence curve of PROSSTT has positive slope and is consistently above 1, while the Splatter curve has a (clear) negative slope and stays below 1.

After quantifying divergence in both PROSSTT and Splatter simulations, we conclude that the simulations produced by Splatter have inherent characteristics that prevent algorithms that try to find a global structure (like MERLoT and Monocle2) from reaching their full potential. Consequently, we decided to only use simpler topologies for the Splatter benchmark (up to 4 bifurcations), where the impact of short-circuits was less dominant.

Suppl. Note 4: Downstream analysis

Differentially expressed genes detection

After a lineage tree reconstruction has been performed, MERLoT can easily find groups of genes being differentially expressed among different groups of cells. If two groups of cells are provided, e.g cells assigned to two branches in the tree (Fig. 3C)), MERLoT performs a Kruskal-Wallis rank sum test [14] to evaluate which genes in the full expression matrix are differentially expressed on them. If a single subpopulation of cells is provided, the comparison is made against the rest of cells in the data. The entire list of genes is given as output, ordered by the test p-values results. Also, e-values are provided by multiplying the p-values by the number of G genes being tested.

Kruskal-Wallis rank sum test:
Also known as the Kruskal-Wallis one-way analysis of variance, the Kruskal-Wallis rank sum test [14] is a non-parametric method for testing whether the underlying distribution of C samples is identical. The observations are ranked across samples, and a test statistic H is calculated to compare the average rank within each group to the overall average rank. The intuition behind the test is that if samples are identically distributed, then their average ranks will not differ significantly, and the value of the test statistic will be maximized.

Suppl. Note 4:.1 GCN reconstruction

MERLoT can be used to study gene-gene expression correlation along every tree branch. In a proof-of-concept we have illustrated how this can be exploited to derive a gene-gene correlation network. In the example showcased in this paper (see Fig. 8 and Figs. S10-S13) genes with similar functions, as characterized by their Gene Ontology annotation, clustered together. Whole clusters were upregulated or downregulated in different parts of the tree, reflecting the multiple differentiation stages and paths available in the process.

Suppl. Note 5: GCNs for Guo 2010

When analyzing datasets containing more than 2 populations of cell types there are more chances to be affected by the Simpson's Paradox, i.e having spurious gene-gene correlations due to an artifact that occurs when taking all cells together in contrast to the correlations that can be recovered when considering subpopulations of specific cell types [15]. The Guo dataset contains a population of progenitors, zygote cells, and 3 mature cell types, TE, PE and EPI cell types. After

reconstructing the lineage tree for this dataset (Fig. 2B) and cells are assigned to the different support nodes in the tree structure we can subdivide them in a trajectory-wise fashion.

Starting from the zygote endpoint we can follow paths to the mature cell types endpoints and define 3 different trajectories: TE, PE and EPI. Following the strategy that was used to analyze the Treutlein dataset, in Figs. S20A, B, and C we show the GCN that can be recovered by using all cells together coloring the genes that are differentially expressed in the terminal branches of each trajectory. Significantly upregulated genes are coloured in shades of red, downregulated genes are coloured in shades of blue and non-differentially expressed genes are coloured in black. Genes that are upregulated in the PE (Figs. S20B) and EPI (Figs. S20C) branches are located in the same region of the GCN with many of the genes being shared between the two trajectories. These two populations emerge from the ICM branch and split away in their last branches. As it was shown in Fig. 3 some genes are upregulated in the EPI branch and downregulated in the PE branch or vice versa. If all cells are taken at the same time to calculate the correlation between pairs of genes, the obtained values can be misleading due to the averaging between the two trajectories.

In Figs. S20D, E and F we show the GCNs that are recovered by using only those cells that lie in the trajectory that goes from the zygote subpopulation of cells to each of the mature cell types. By doing this, we overcome the Simpson's paradox and recover more accurate correlation values for a given trajectory. In the GCNs using all cells, genes like Dab2 are close to the cluster of highly expressed genes regardless of it being upregulated in the PE branch and neutral in the EPI branch. In contrast, we observe that Dab2 appears as a hub node close to the cluster of highly expressed genes in the PE trajectory GCN, while it locates itself close to other neutral genes in the EPI GCN. The more branches a lineage tree contains the more important it becomes to correctly separate multiple trajectories for downstream analysis like GCN or GRN reconstruction.

Suppl. Note 6: MERLoT in $\mathbb{R}^D, D > 3$

In the original Monocle2 paper [16], the authors show in Supplementary Figure 16 their analysis of the haematopoiesis dataset produced by Paul *et al.* [3]. They use 10 components of the DDRTree projection to recover a topology with 5 branching points. We followed this analysis and obtained the DDRTree coordinates. We fitted a scaffold tree and an elastic tree ("auto" mode, with local averaging to 800 cells, and minimum sensitive branch length $\sqrt{N}/2$) on these coordinates and visualized the resulting tree using as annotation the cell types Paul *et al.* assigned to the various clusters.

Monocle2 does a good job of separating the multiple cell fates. On the top of the tree (Fig. S21A) the first branch is composed of multipotent progenitors (in cyan), mixed with more mature cells. This branch separates into the erythroid lineage (purple), the megacaryocyte branch (blue) and the granulocyte/monocyte progenitor (brown) branch, which gives rise to the (mostly) basophil branch as well as three other mixed branches: one with a mixture of dendritic cells (red) and monocytes (orange), one that contains cells from different groups, and one predominantly neutrophilic (magenta)/monocytic branch.

MERLoT however improves quite a bit on this (Fig. S21C and D). In particular, MERLoT separates the megakaryocyte and the dendritic cell lineages much more clearly, despite them being underrepresented (S21C). The erythrocyte branch reflects the Monocle2 assignment, with most cells further away from the tip, while the granulocyte/monocyte internal branch has multiple basophil cells, much like Monocle2. An interesting detail is that while MERLoT also mixes multiple mature cell types with the multipotent progenitors, it does a better job of separating them, even suggesting another sparsely populated branch for mostly neutrophil cells. Finally, MERLoT proposes a split for the common monocyte/neutrophil lineage.

The analysis and plotting can be found in `https://github.com/soedinglab/merlot-scripts`.

Data availability

The 10 simulation sets with 100 simulated differentiations each are available at `http://wwwuser.gwdg.de/~{}compbiol/merlot/`. The code necessary to run the benchmark on the simulations as well as instructions about how to set up a similar benchmark are available at `https://github.com/soedinglab/merlot-scripts`. Formatted expression data for the three datasets in Fig. 2 are available at: `https://github.com/soedinglab/merlot/tree/master/inst/example/`.

Competing interests

The authors declare that they have no competing interests.

Supplementary Figures

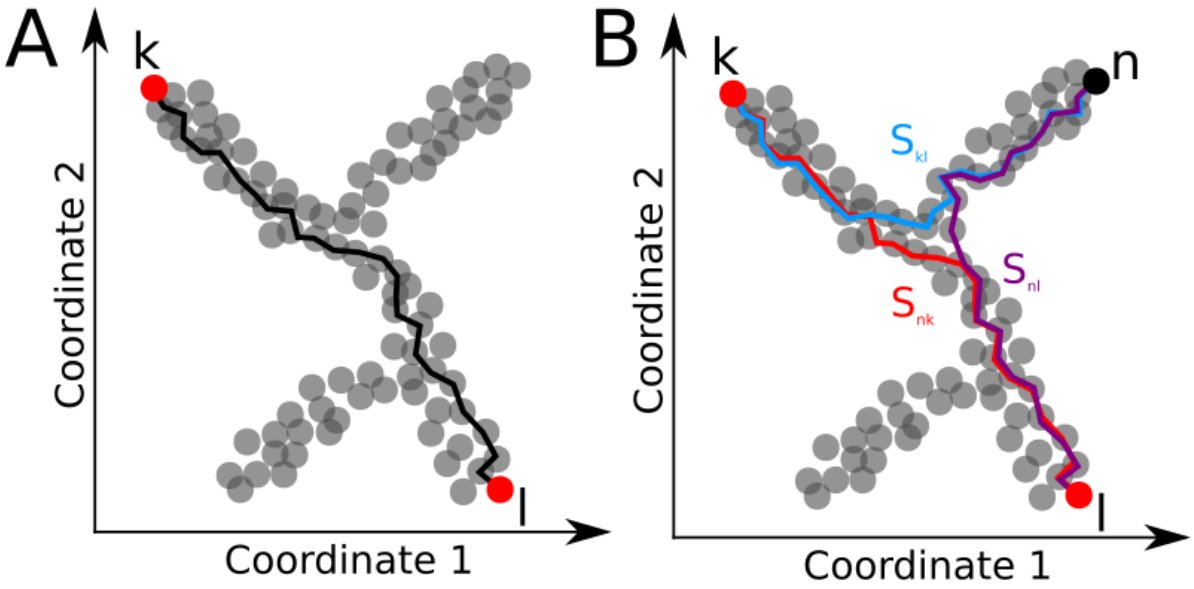

Figure S1: MERLoT's method for finding endpoints: (A) Pair of cells that maximize the length of the shortest path distances matrix in the dataset. **(B)** Every cell n, not in the shortest path between already found endpoints is evaluated according to: $s_{\mathcal{E}}(n) := 0.5 \times \min\{S_{kn} + S_{nl} - S_{kl} \ : \ k, l \in \mathcal{E}\}$. S_ij represents the number of nodes in the shortest path between i, j. The new endpoint to be added to the tree structure is the one that maximizes its $s_{\mathcal{E}}(n)$ value respect to the other cells in the dataset.

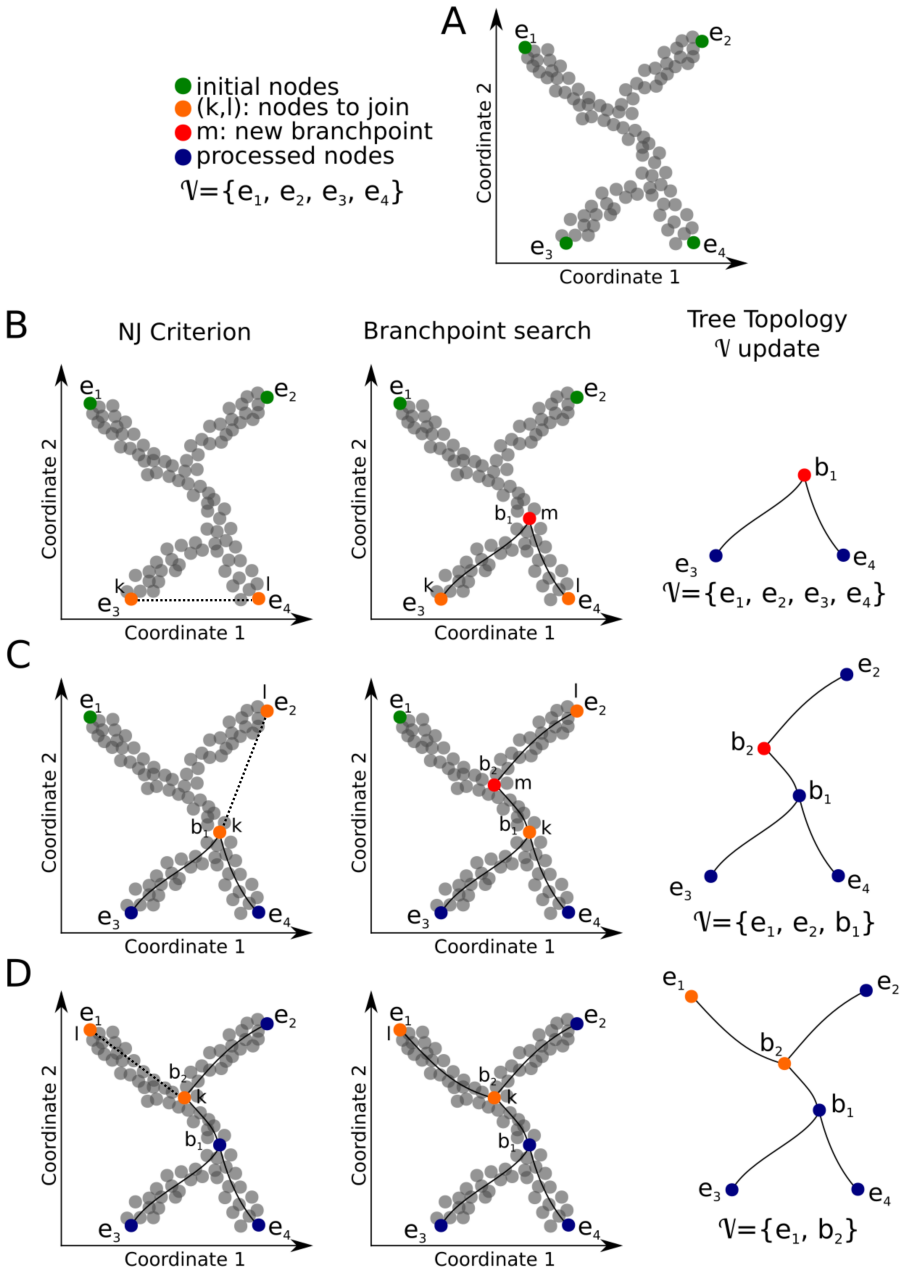

Figure S2: MERLoT's method for finding branchpoints and tree connectivity: The algorithm has two steps (1) Neighbour Joining (NJ) Criterion and (2) branchpoint search. The NJ criterion is applied over the $\mathcal{V}$ vector which is initially set to the set of endpoints (green). In every NJ iteration, two k, l nodes are selected to be joined (dashed lines) for which a branchpoint m is found (red) and the edges between the branchpoint and the joined nodes are added to the topology. The $\mathcal{V}$ vector is updated in every iteration by subtracting the k and l nodes and adding the m node. The procedure stops when $|\mathcal{V}|$=2. **(A)** $\mathcal{V}$ is initialized with the endpoints e1-e4. **(B)** First iteration: e3 and e4 and joined through b1. e3-b1 and e4-b1 edges are added to the tree. e3 and e4 are deleted from $\mathcal{V}$ and b1 is added instead. **(C)** Second iteration: e2 and b1 and joined through b2. e2-b2 and b1-b2 edges are added to the tree. e2 and b1 are deleted from $\mathcal{V}$ and b2 is added instead. **(D)** Third iteration: $|\mathcal{V}|$=2, procedure stops and the b2-e1 edge is added to the tree.

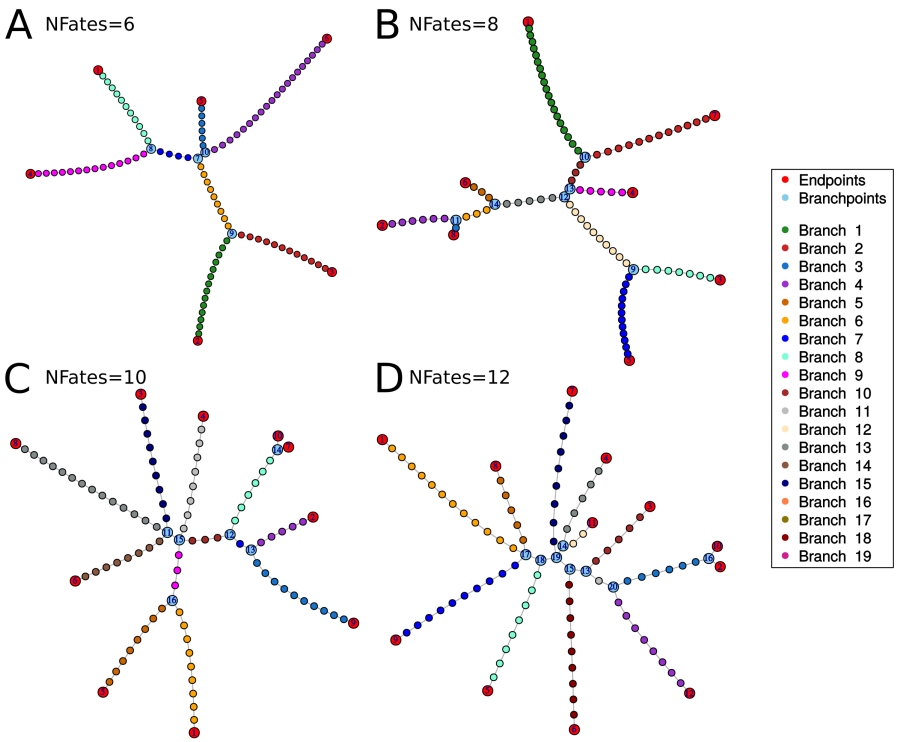

Figure S3: Topology plots: In cases where the dimensionality reduction happens in more than three dimensions, MERLoT can plot a schematic representation of the lineage tree.

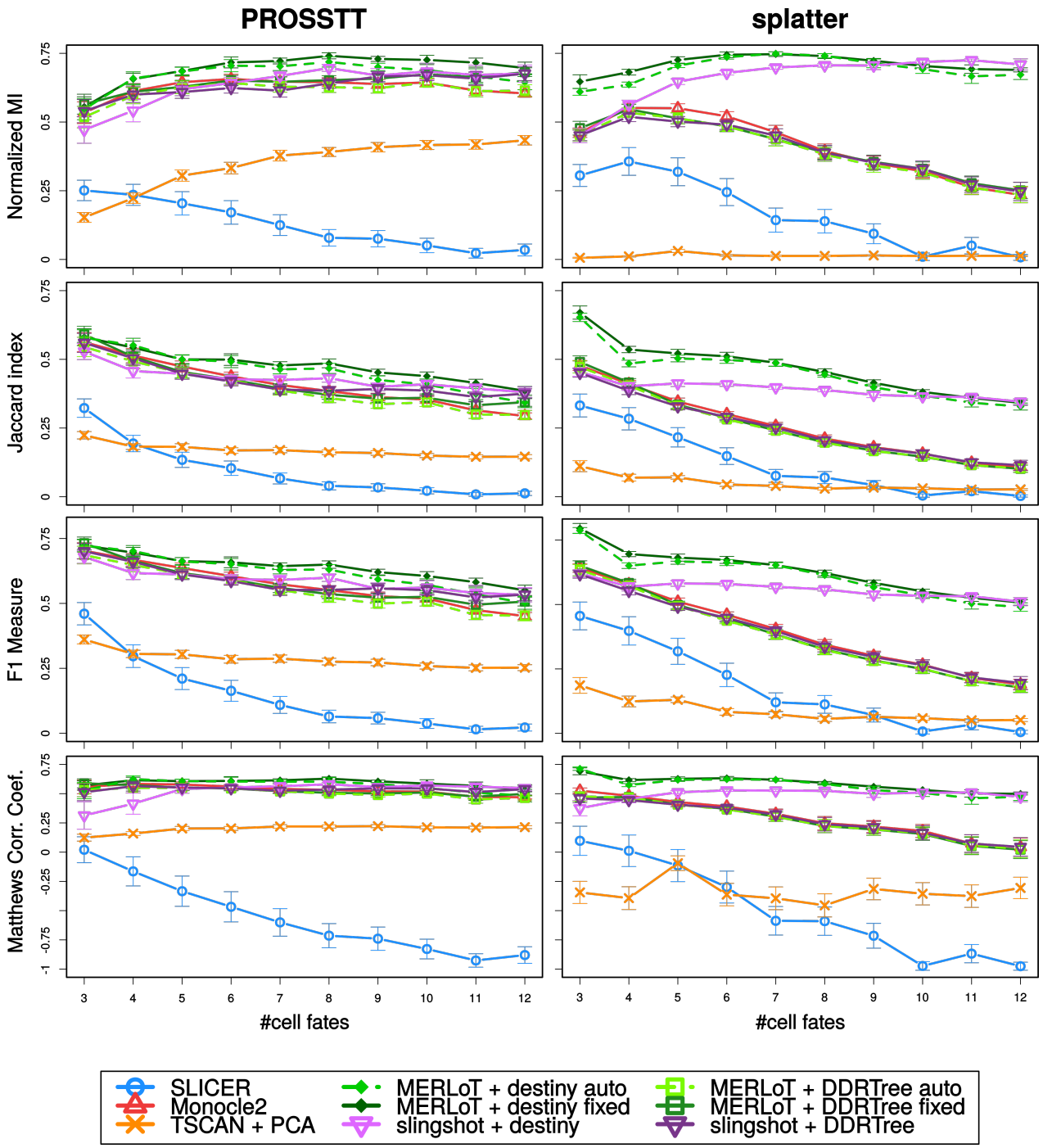

Figure S4: Branch assignment benchmarking. We calculated different scores to assess how good the different methods are at assigning cells to the different measures on the PROSSTT (left) and Splatter (right) simulation sets. From top to bottom, NMI, Jaccard Index, F1 measure, and MCC. The error bars are 95% confidence intervals assuming the prediction scores are normally distributed.

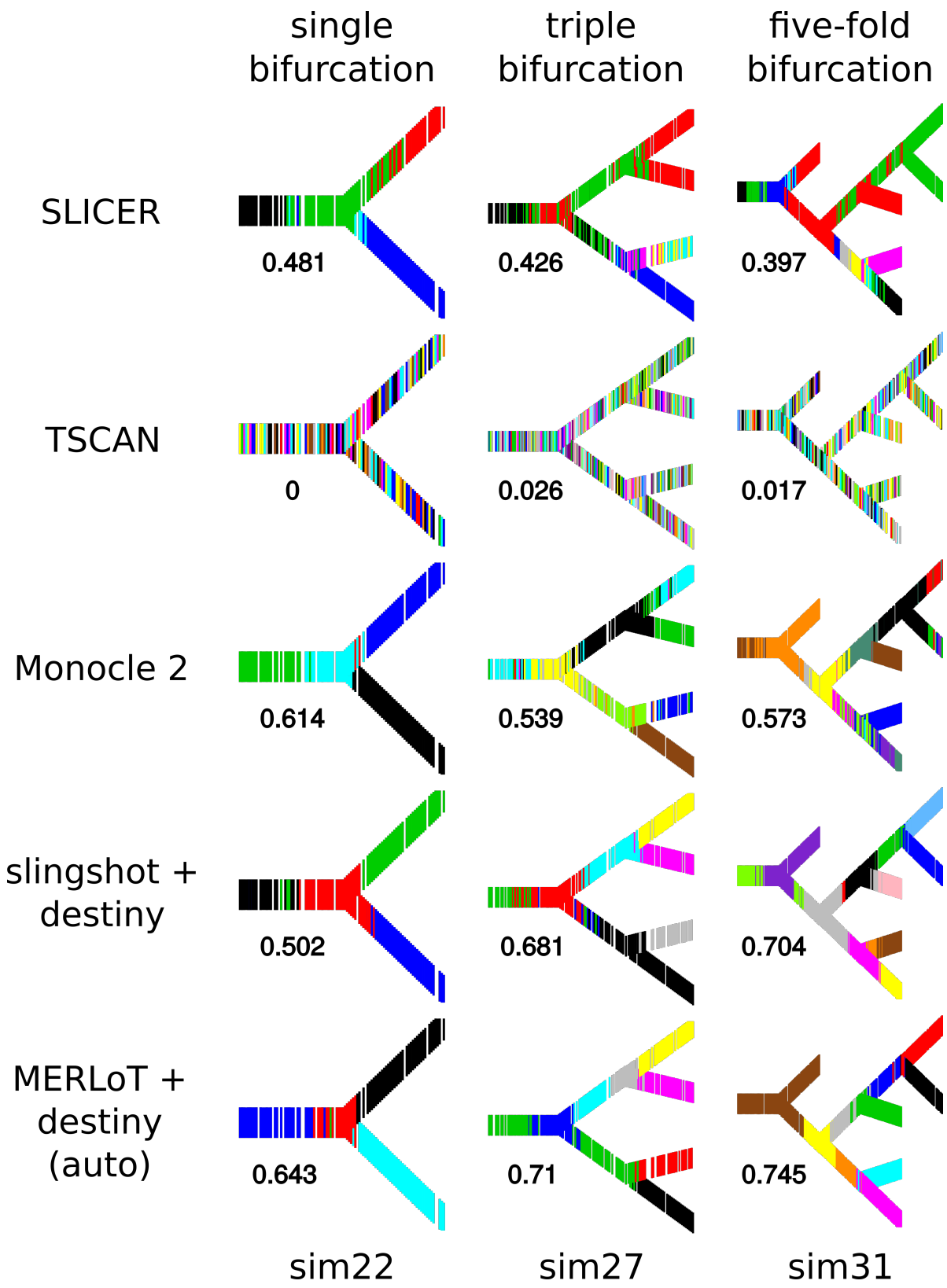

Figure S5: Examples of branch assignments for MERLoT, Monocle2, and SLICER. We show three typical examples from the single (left), triple (middle) and five-fold (right) bifurcation benchmark sets (simulation names at bottom). In each panel we show the lineage tree of the simulated tree, with vertical lines representing the cells, ordered by pseudotime. The color of each line is the branch label predicted for it by the respective method. The NMI of each prediction is in the top left of each panel. We picked examples with NMI values near the average values for each tool and tree topology.

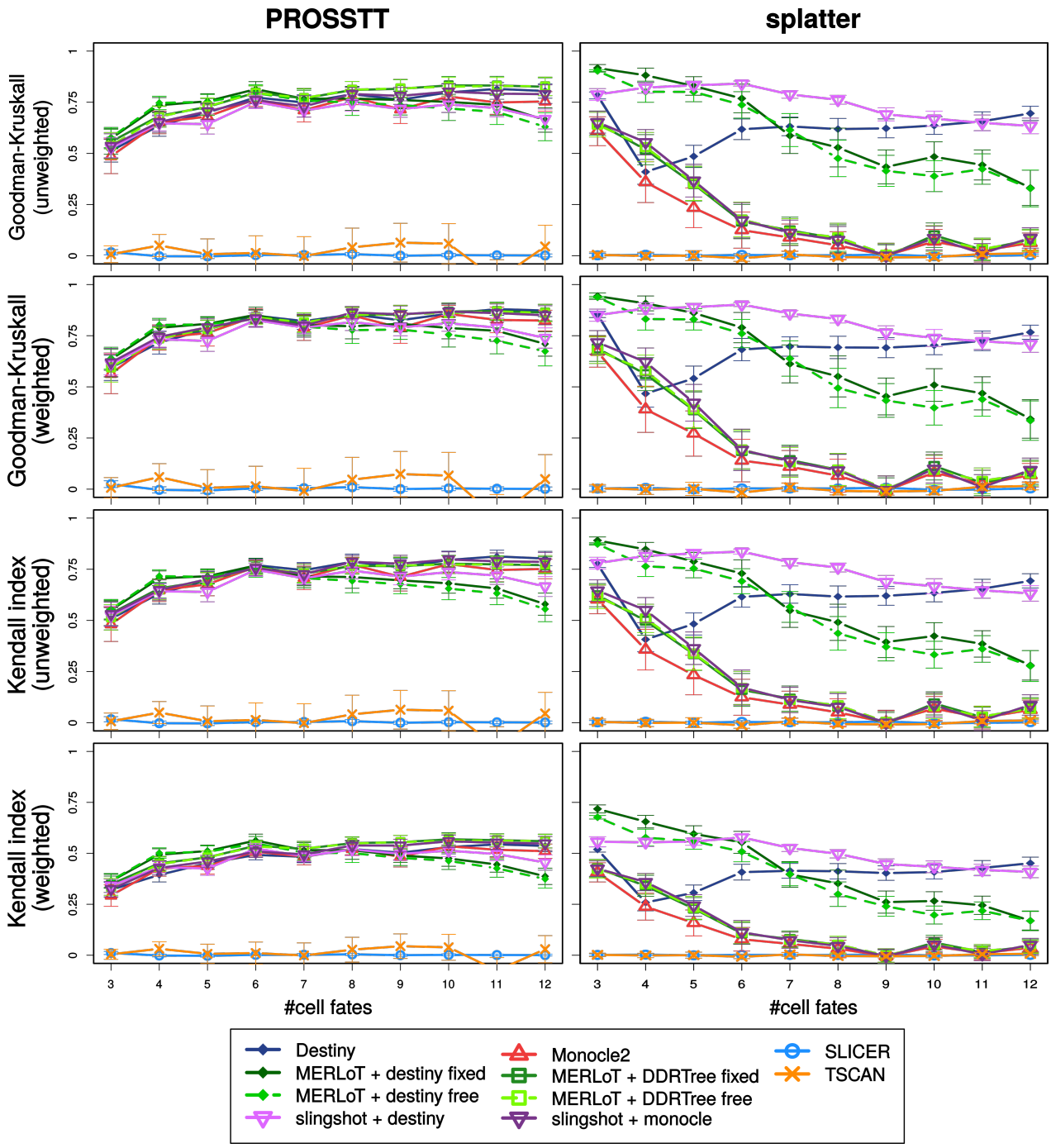

Figure S6: Pseudotime assignment benchmarking. We calculated different scores to assess how good the different methods are at assigning pseudotime values to cells along the longest sub tree in the PROSSTT (left) and Splatter (right) simulation sets. From top to bottom, Goodman-Kruskal index (unweighted and weighted), and Kendall index (unweighted and weighted). The error bars are 95% confidence intervals assuming the prediction scores are normally distributed.

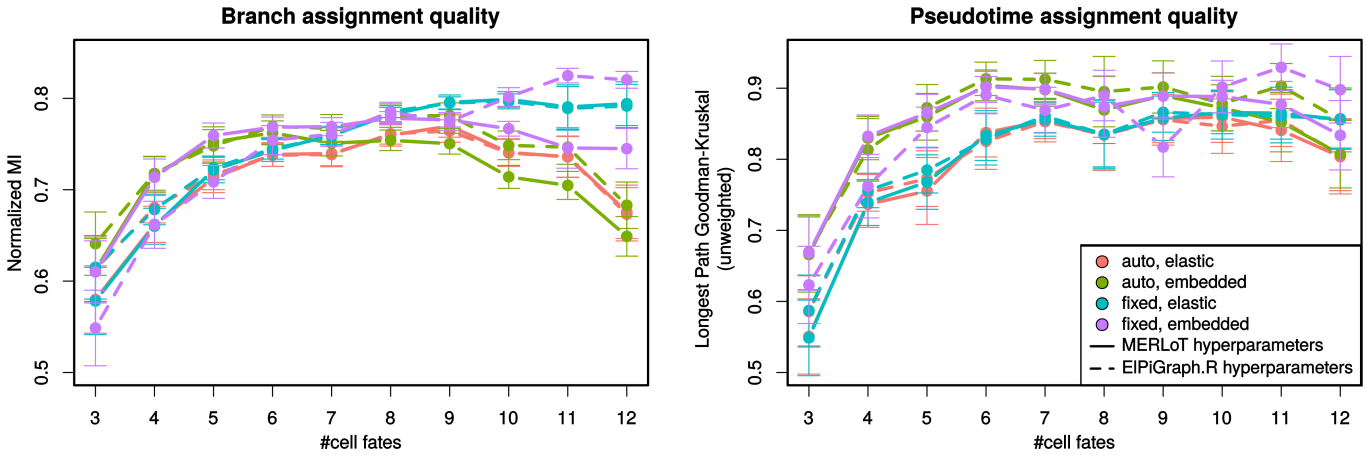

Figure S7: Comparison of hyperparameter performance: We compared the average performance of different MERLoT flavours in vanilla mode against MERLoT with ElPiGraph.R elasticity hyperparameters. The comparison is on the extended benchmark. The different colours are different MERLoT flavours; solid lines denote MERLoT with MERLoT hyperparameters; dashed lines denote MERLoT with ElPiGraph.R hyperparameters. The error bars are 95% confidence intervals assuming the prediction scores are normally distributed. All MERLoT flavours were ran on local averaging mode, as was the whole benchmark.

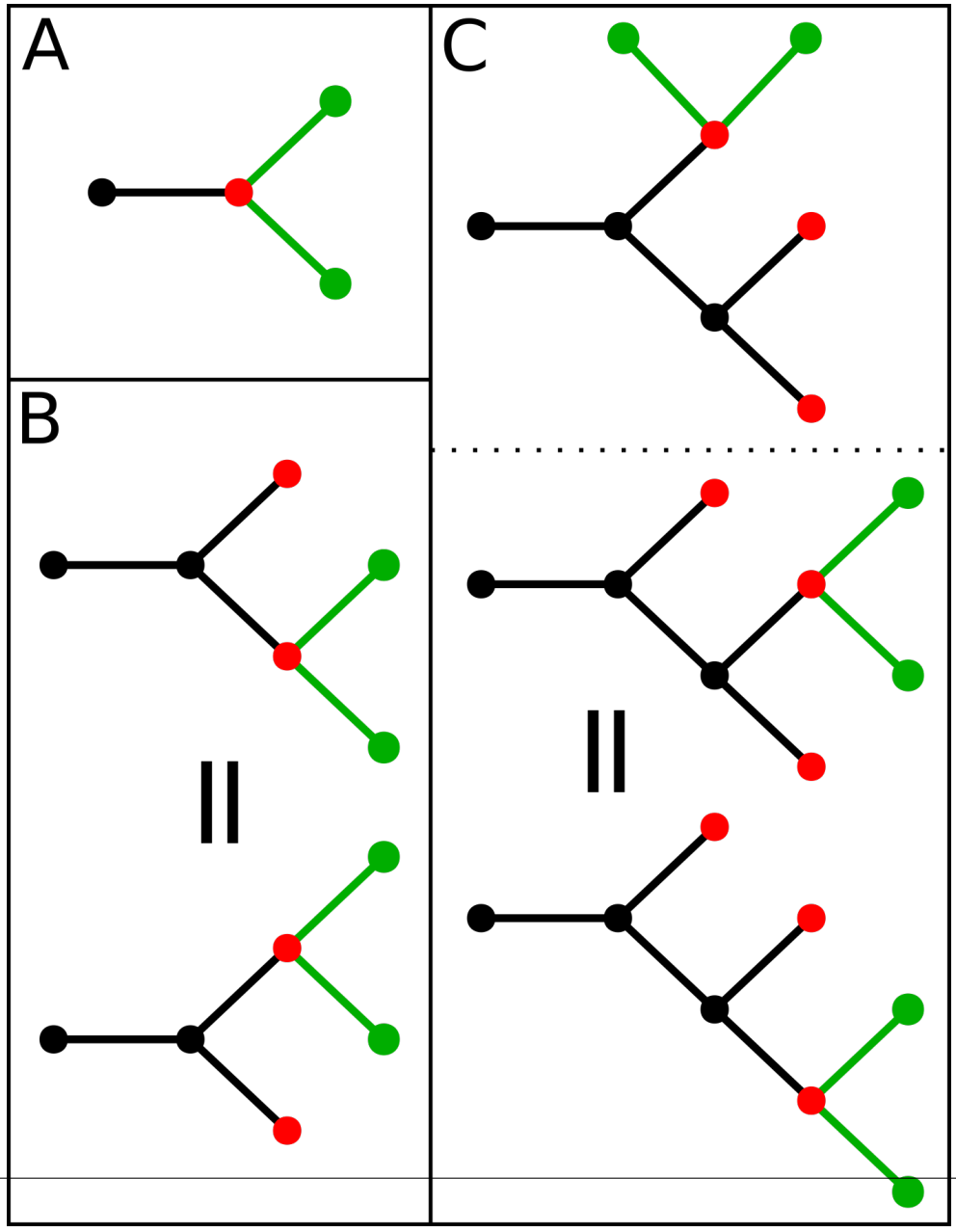

Figure S8: Topology Complexity. **(A)** With one bifurcation only one topology is possible. **(B)** With two bifurcations two topologies are possible. However, they are equivalent in terms of length of the longest subtree. **(C)** More topologies are possible. For the upper case in panel B, three more topologies can be constructed by adding a new bifurcation. Not all topologies are equivalent in terms of the length of the longest sub tree.

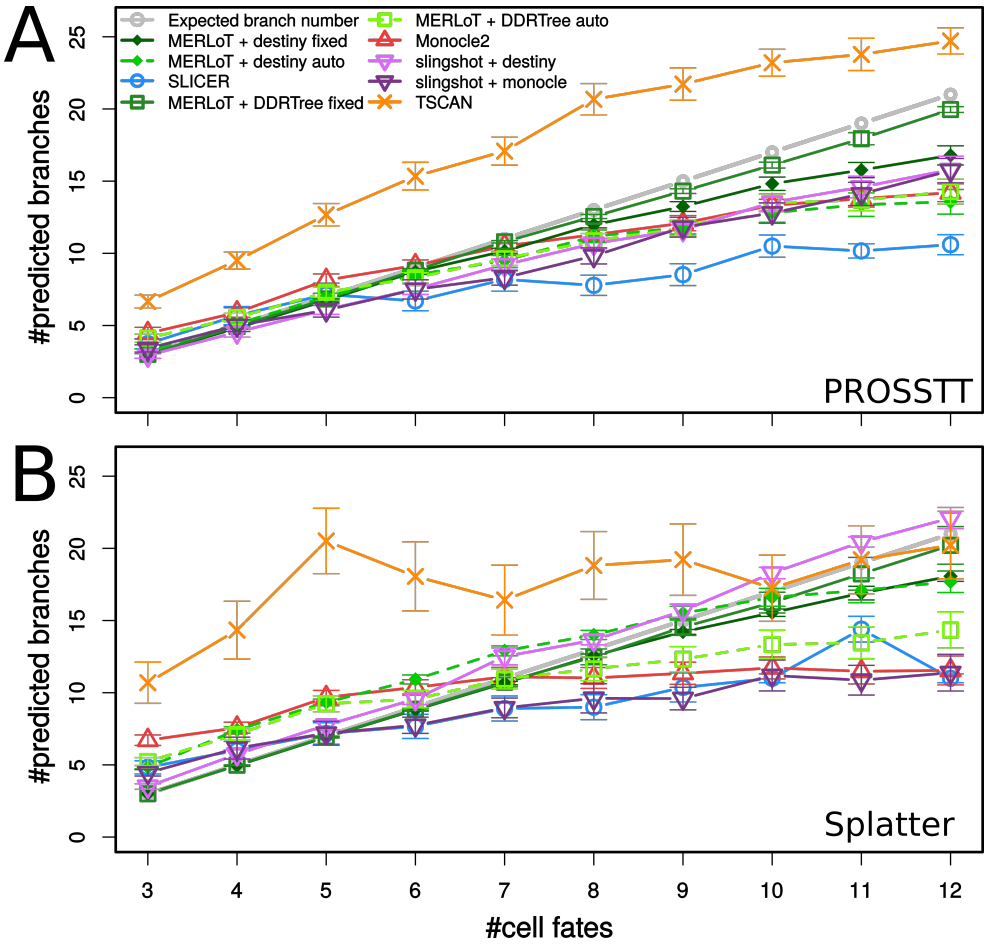

Figure S9: Number of predicted branches in simulated datasets. We compare how many branches out of the true number of generated branches each tool is able to predict. The true simulated number of branches y shown in gray, and the different tools are shown in different colors and symbols as described in the legend. **(A)** PROSSTT set. **(B)** Splatter set. The error bars are 95% confidence intervals assuming the prediction scores are normally distributed.

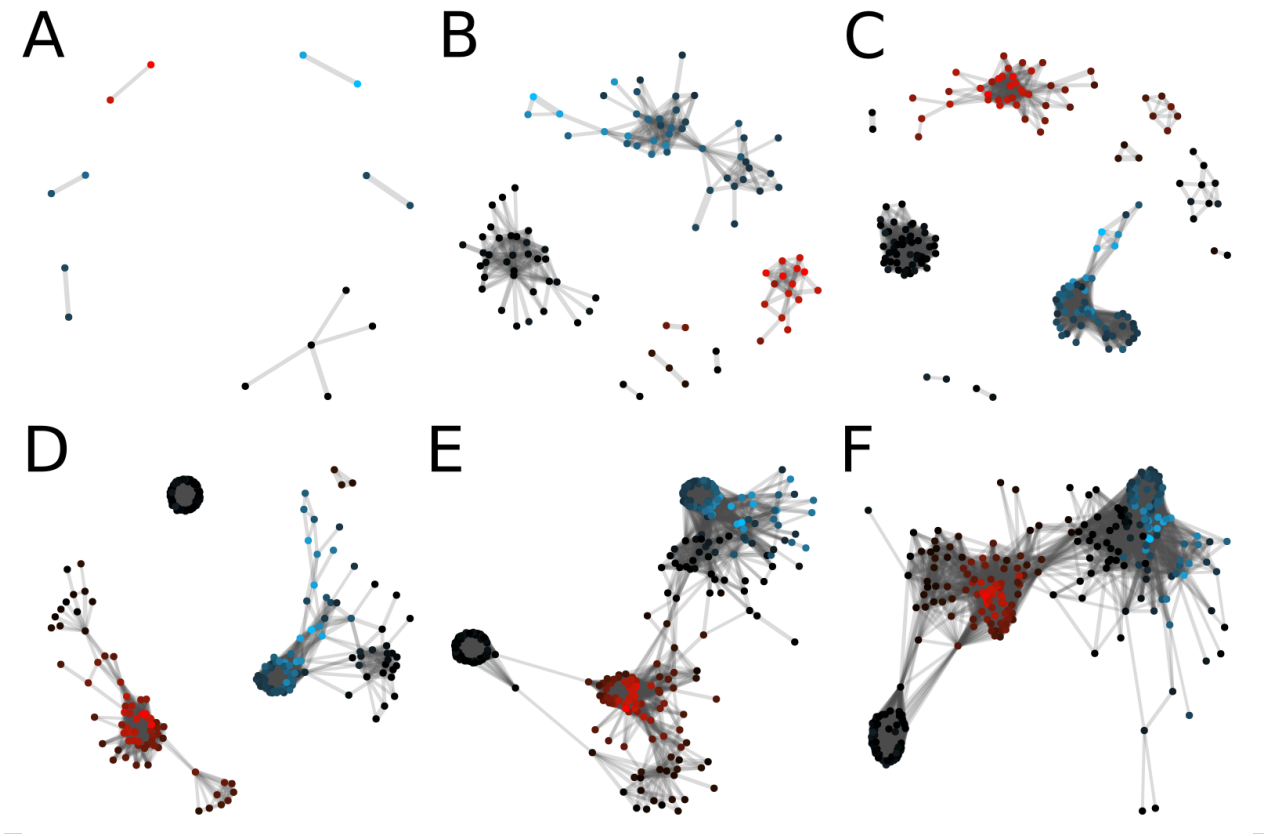

Figure S10: Fibroblasts differentially expressed genes mapped in the GCN structure. GCN is reconstructed using different thresholds for Pearson's correlation coefficient between genes: **(A)** 0.9, **(B)** 0.8, **(C)** 0.7, **(D)** 0.6, **(E)** 0.5, **(F)** 0.4. Genes in red are significantly upregulated. Genes in blue are significantly downregulated. Genes in black are not significantly differentially expressed.

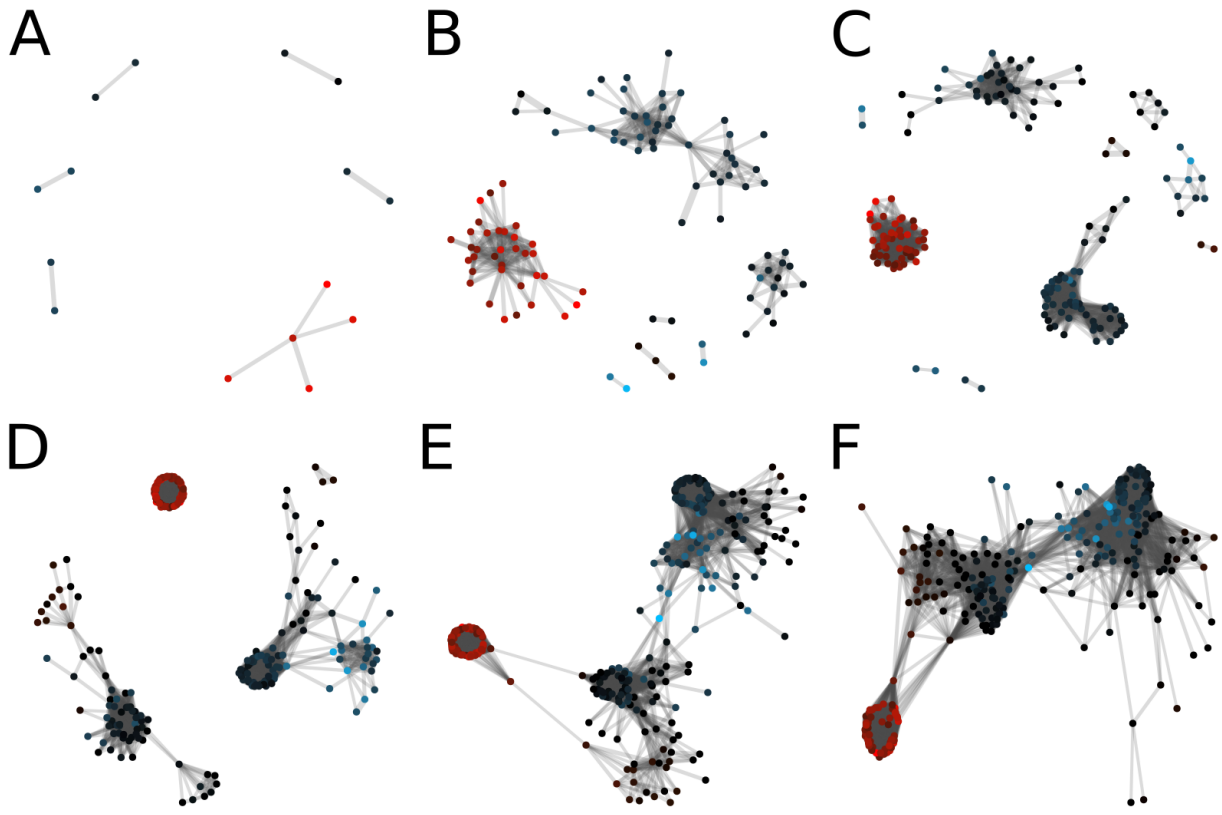

Figure S11: Neurons differentially expressed genes mapped in the GCN structure. GCN is reconstructed using different thresholds for Pearson's correlation coefficient between genes: **(A)** 0.9, **(B)** 0.8, **(C)** 0.7, **(D)** 0.6, **(E)** 0.5, **(F)** 0.4. Genes in red are significantly upregulated. Genes in blue are significantly downregulated. Genes in black are not significantly differentially expressed.

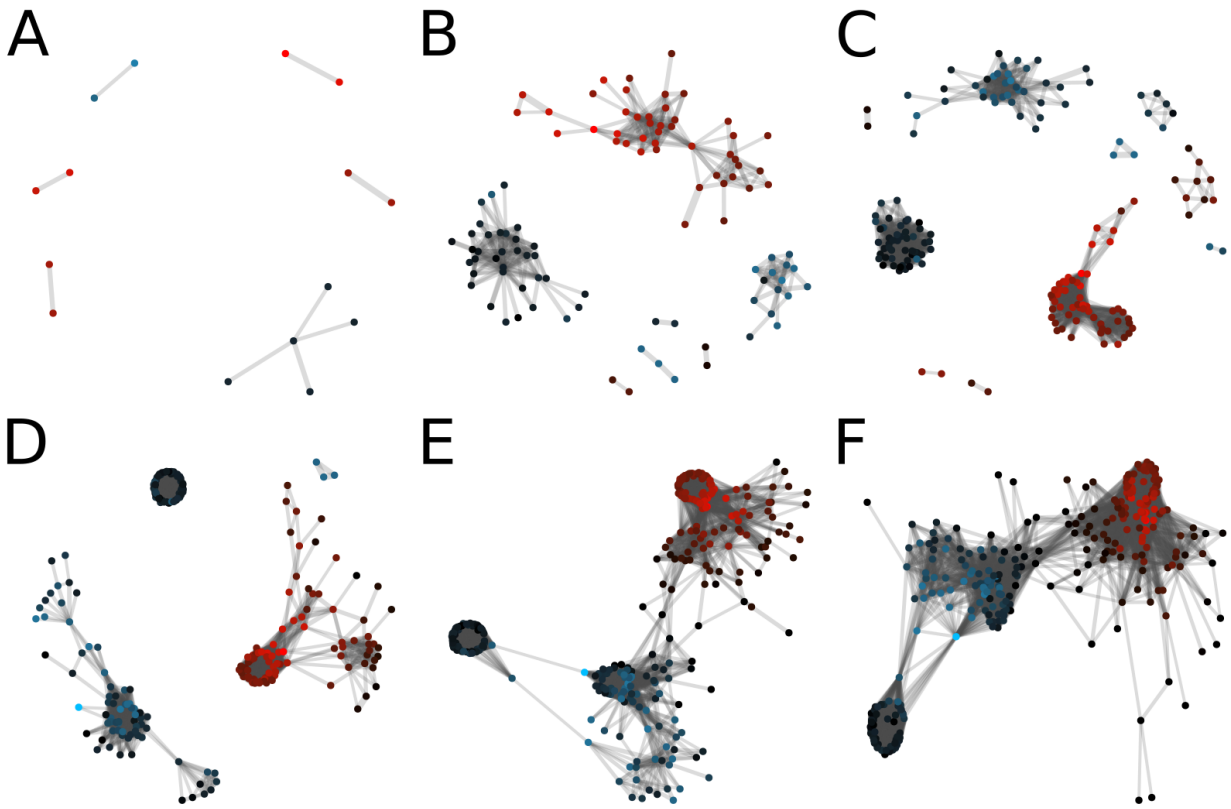

Figure S12: Myocytes differentially expressed genes mapped in the GCN structure. GCN is reconstructed using different thresholds for Pearson's correlation coefficient between genes: **(A)** 0.9, **(B)** 0.8, **(C)** 0.7, **(D)** 0.6, **(E)** 0.5, **(F)** 0.4. Genes in red are significantly upregulated. Genes in blue are significantly downregulated. Genes in black are not significantly differentially expressed.

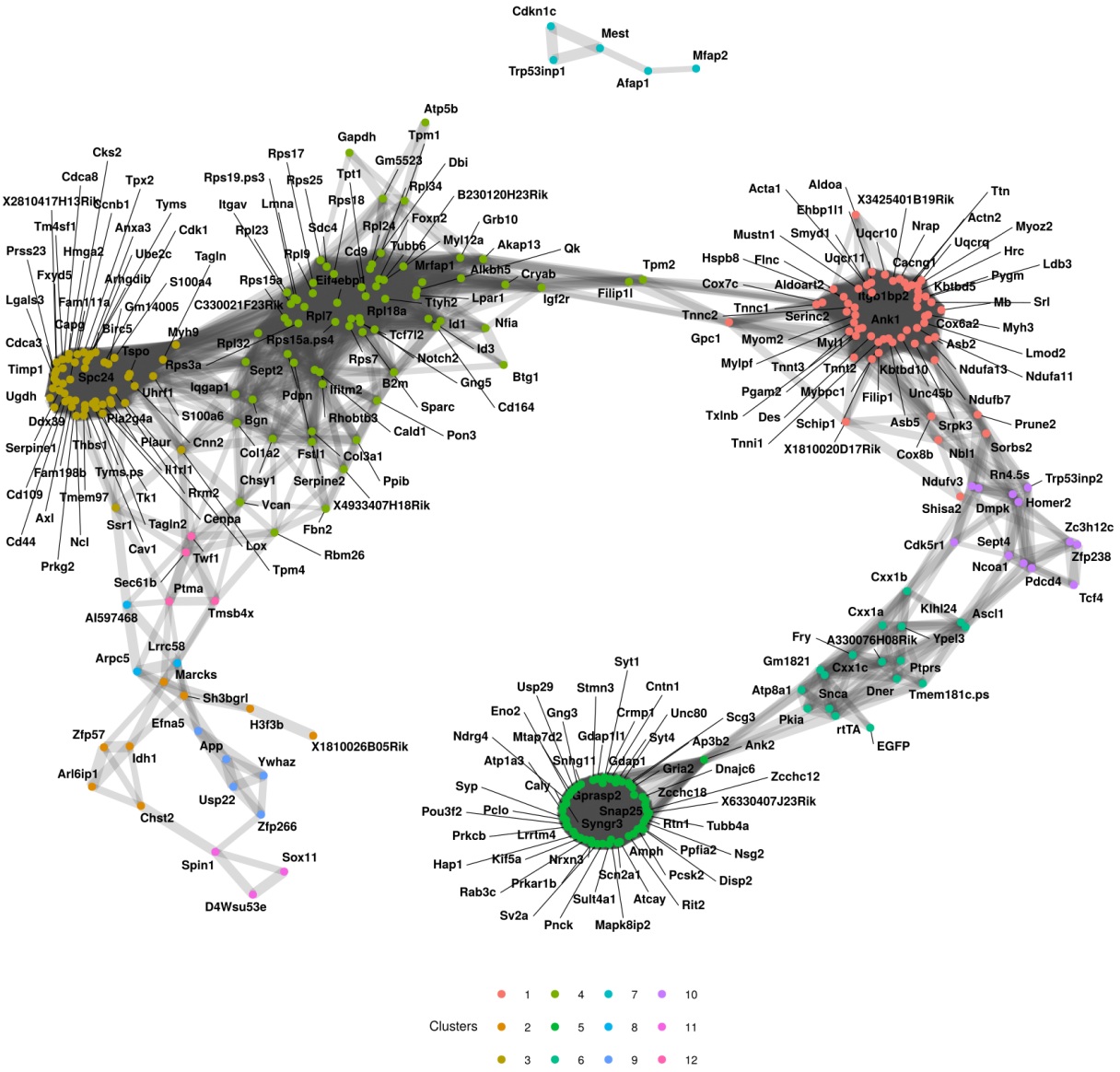

Figure S13: Labeled GCN. Genes were clustered using the "walktrap" algorithm. Groups of genes are coloured according to the clusters found in the network structure.

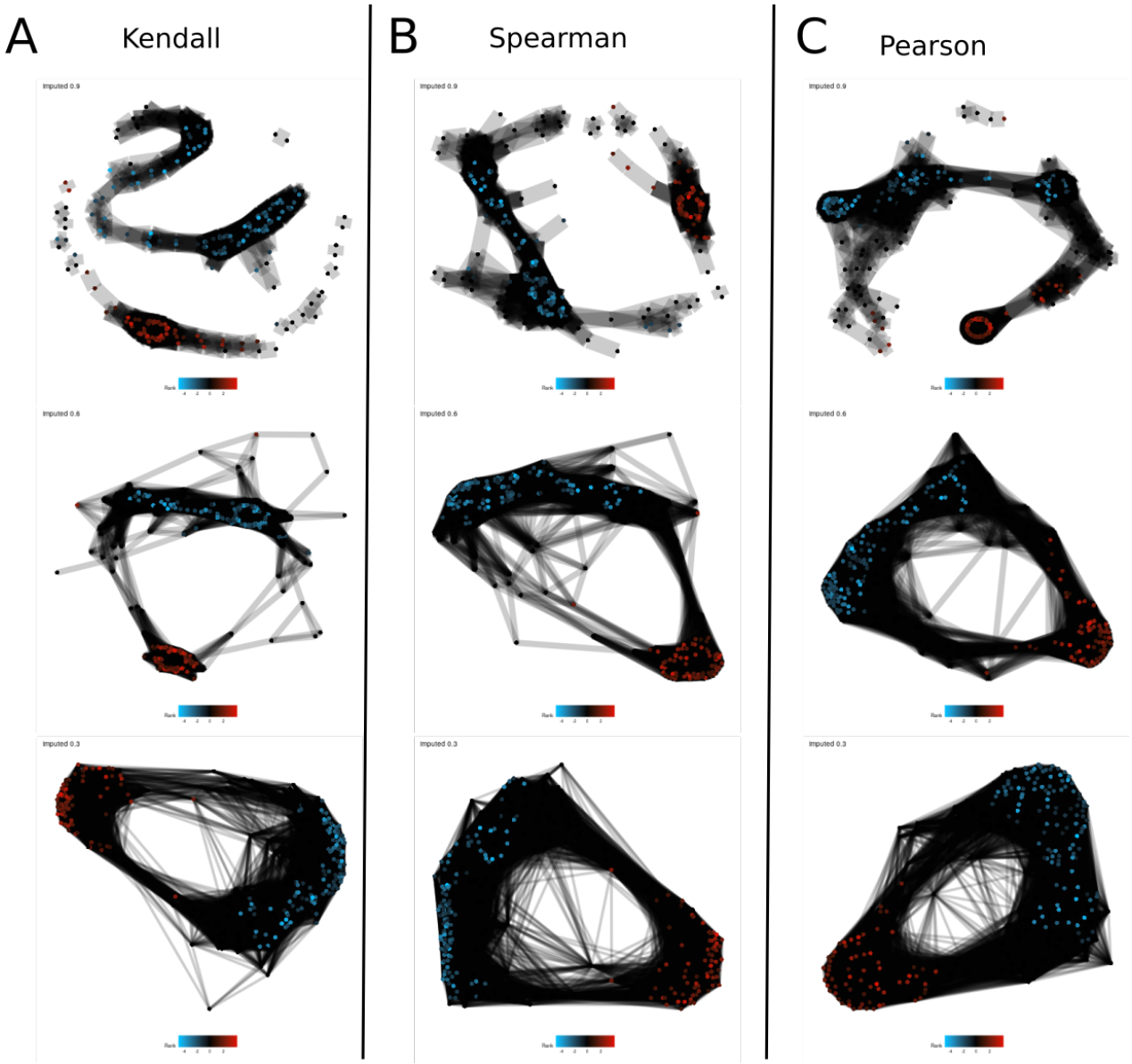

Figure S14: GCN - other measures. Reconstruction of the Treutlein GCN with different correlation measures. **(A)** Kendall's τ coefficient, **(B)** Spearman's ρ, **(C)** Pearson's r. From top to bottom, each row has a different cut-off for the inclusion of edges. Top row is 0.9, middle is 0.6 and bottom is 0.3. The top row of column C is the configuration presented in the main paper. Repeating the GCN reconstruction with different correlation measures produces very similar results, albeit without the clear clustering suggested by the Pearson correlation coefficient analysis. While a threshold of 0.9 was appropriate for the Pearson correlation, it seems as if lower cut-offs might be required for the Spearman and Kendall coefficients.

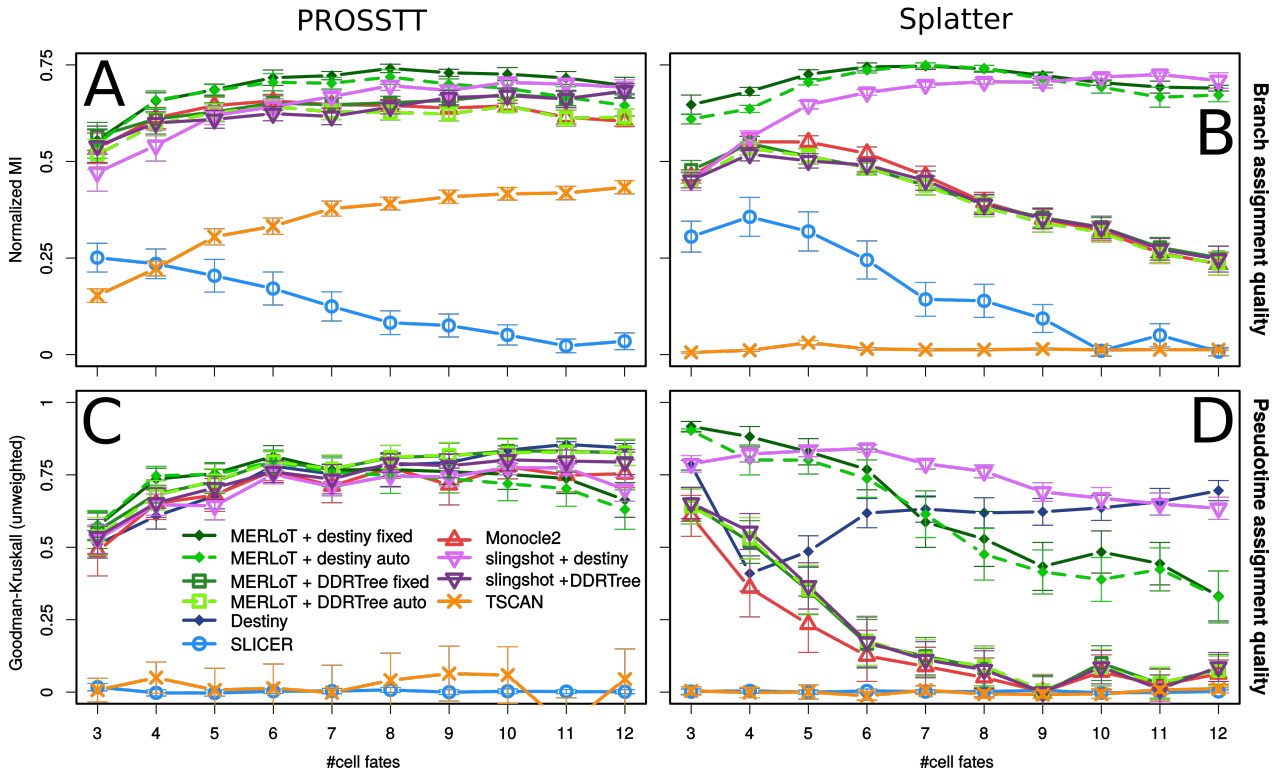

Figure S15: Benchmark results. (A)-(D) Branch assignment (upper panels) and pseudotime assignment (lower panels) comparison using Monocle2, SLICER, TSCAN, Slingshot, and MERLoT using both PROSSTT (left) and Splatter (right) simulations. Slingshot and MERLoT are used in combination with DDRTree and diffusion map (Destiny) coordinates. Compare with Fig. 5. The error bars are 95% confidence intervals assuming the prediction scores are normally distributed.

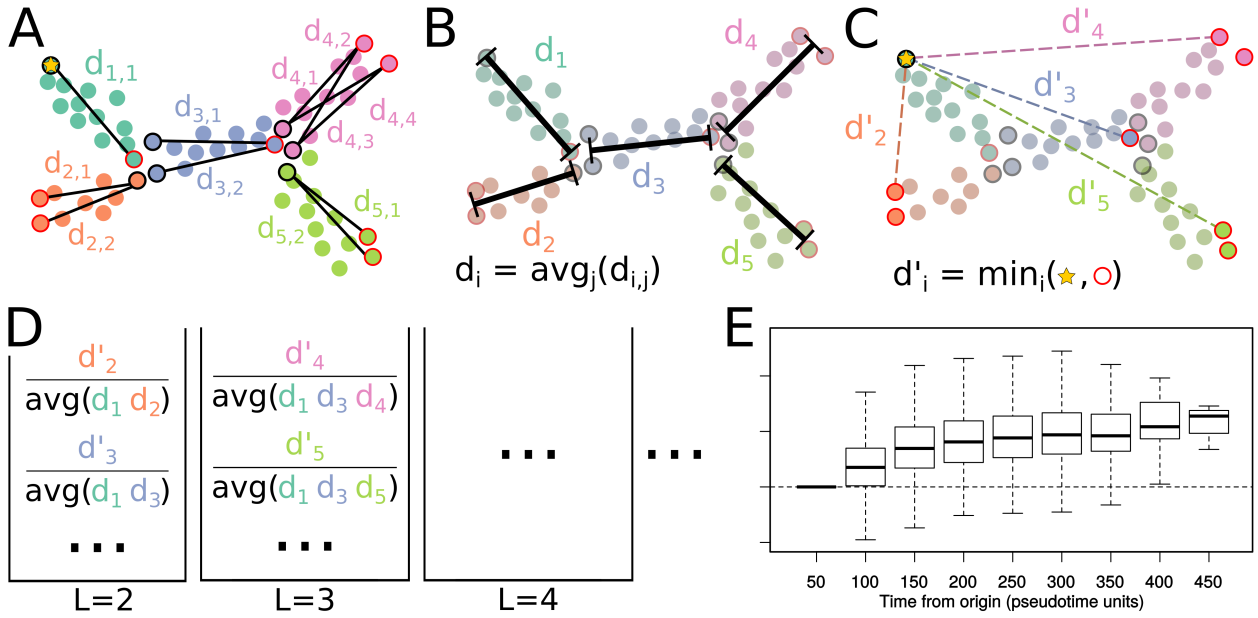

Figure S16: Divergence analysis. (A) We locate the cells with minimum and maximum pseudotime in each branch and calculate their pairwise distances. **(B)** The average of these distances is the branch length. **(C)** We calculate the minimum direct distance of the origin of the differentiation (minimum pseudotime of first branch) to the other waypoints (maximum pseudotime of each branch). **(D)** We take all on-tree paths from the origin to a waypoint and normalize the minimum direct distance via the average branch length in the path. After repeating steps (A)-(C) for all simulations in the set, we group these values by path length (in branches). **(E)** The boxplots of each path bin constitute the divergence curve of the simulations. If the branches all have the same pseudotime length, then distance to waypoints is equivalent to the pseudotime that has passed since the origin.

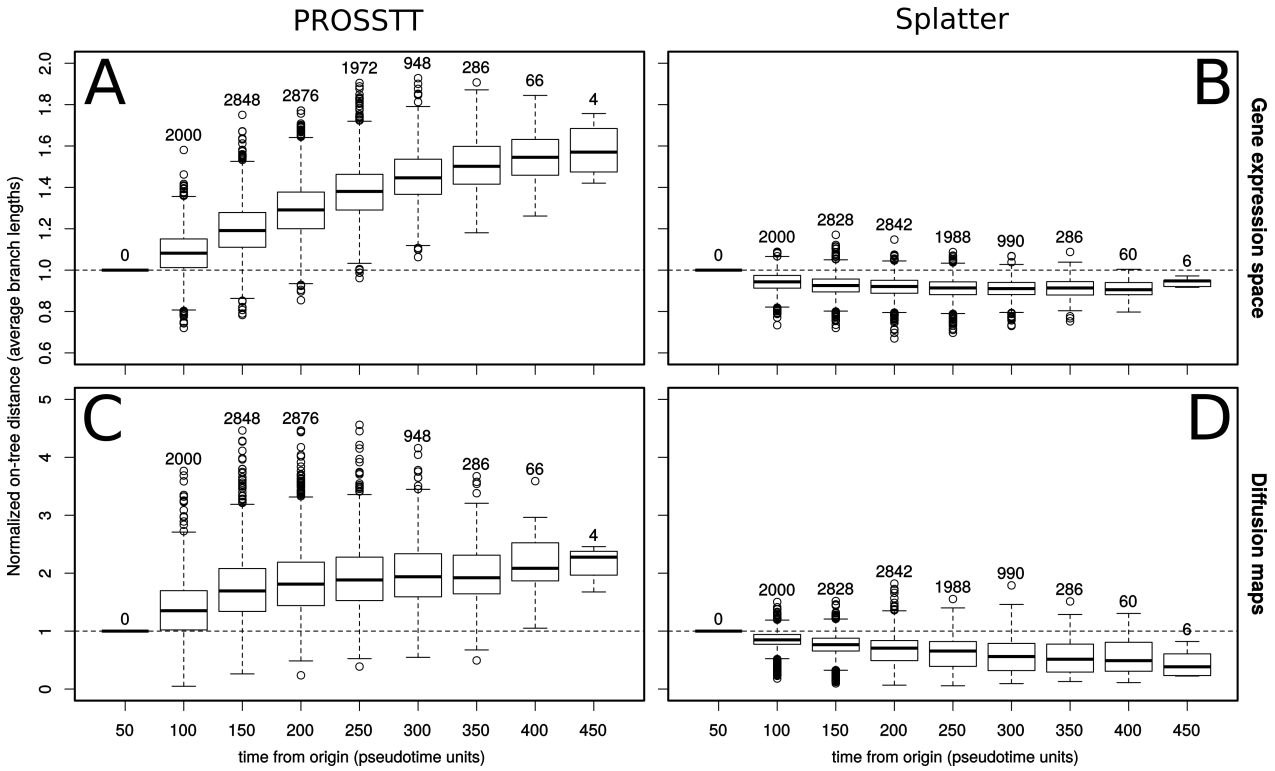

Figure S17: Divergence analysis results. Divergence analysis for PROSSTT (left) and Splatter (right) simulations in gene expression space (upper panels) or in diffusion space (lower panels).

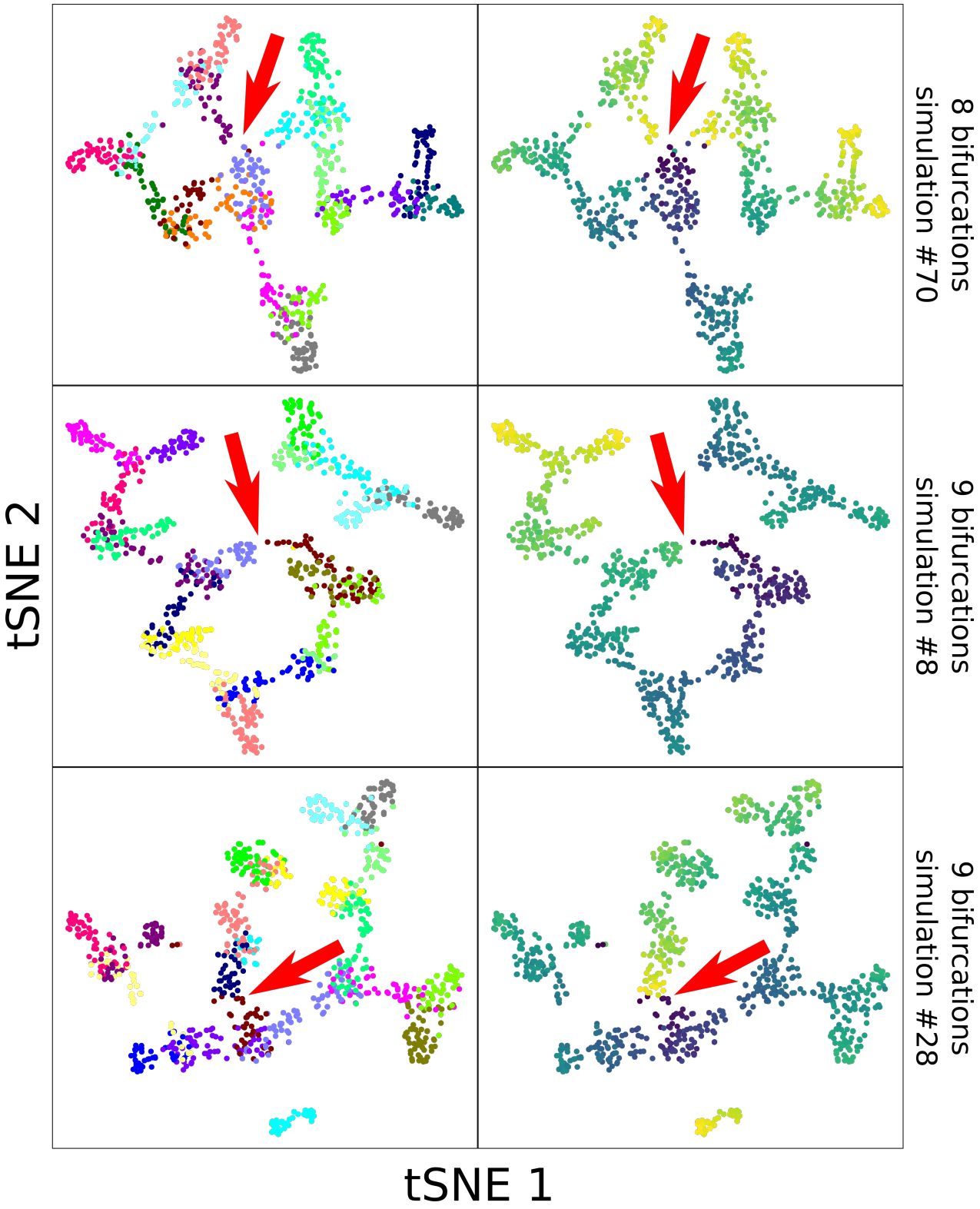

Figure S18: Short circuits. tSNE plots of Splatter simulations that show short-circuits. On the left side the colors denote the branches. On the right side the color denotes the pseudotime, from low (dark blue) to high (yellow). The red arrows point to the exact points where the short-circuit takes place. In all cases, cells with low and high pseudotime are so close to each other that the shape of the lineage tree is not clear, leading to problems in the tree reconstruction.

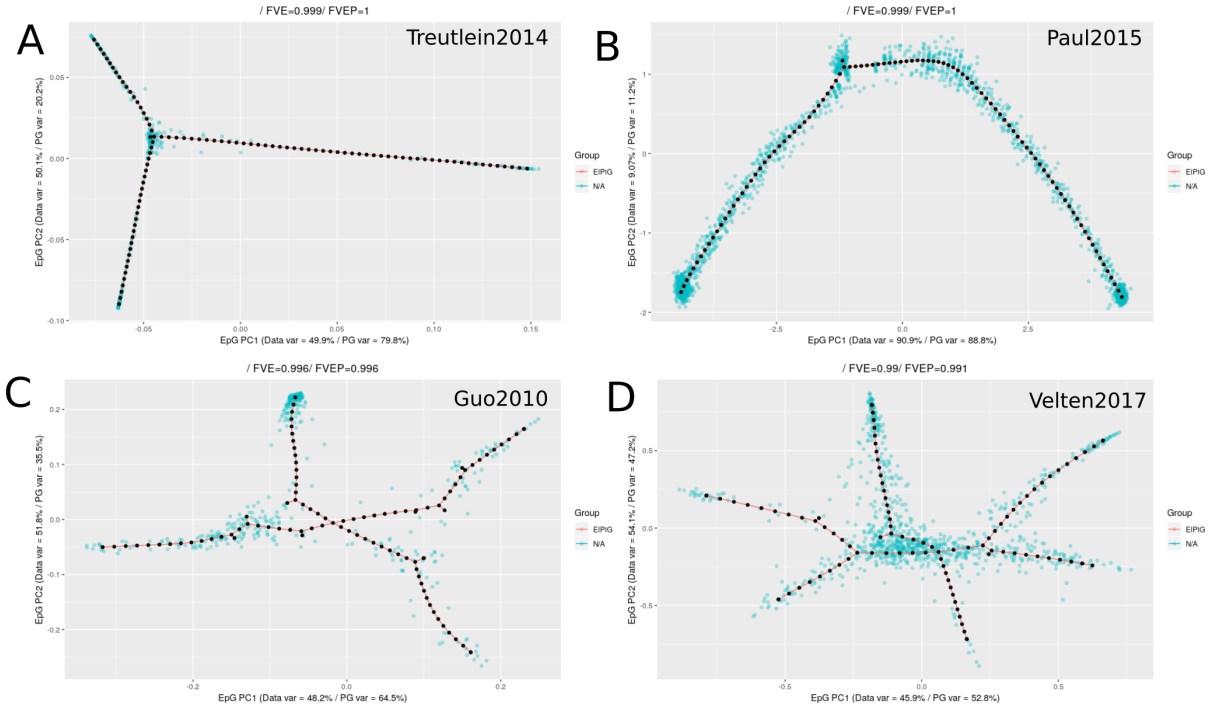

Figure S19: ElPiGraph tree reconstructions. We performed tree reconstructions for the datasets used as examples in the manuscript using the computeElasticPrincipalTree function from ElPiGraph using default parameters. **(A)** Treutlein dataset as used in Fig 5A using diffusion coordinates calculated by the Destiny package. **(B)** Paul dataset as used in Fig. 2D using DDR3 coordinates calculated by the Monocle2 package. **(C)** Guo dataset as used in Fig. 2E using diffusion coordinates calculated by the Destiny package. **(D)** Velten dataset as used in Fig. 2F using coordinates calculated by the STEMNET package.

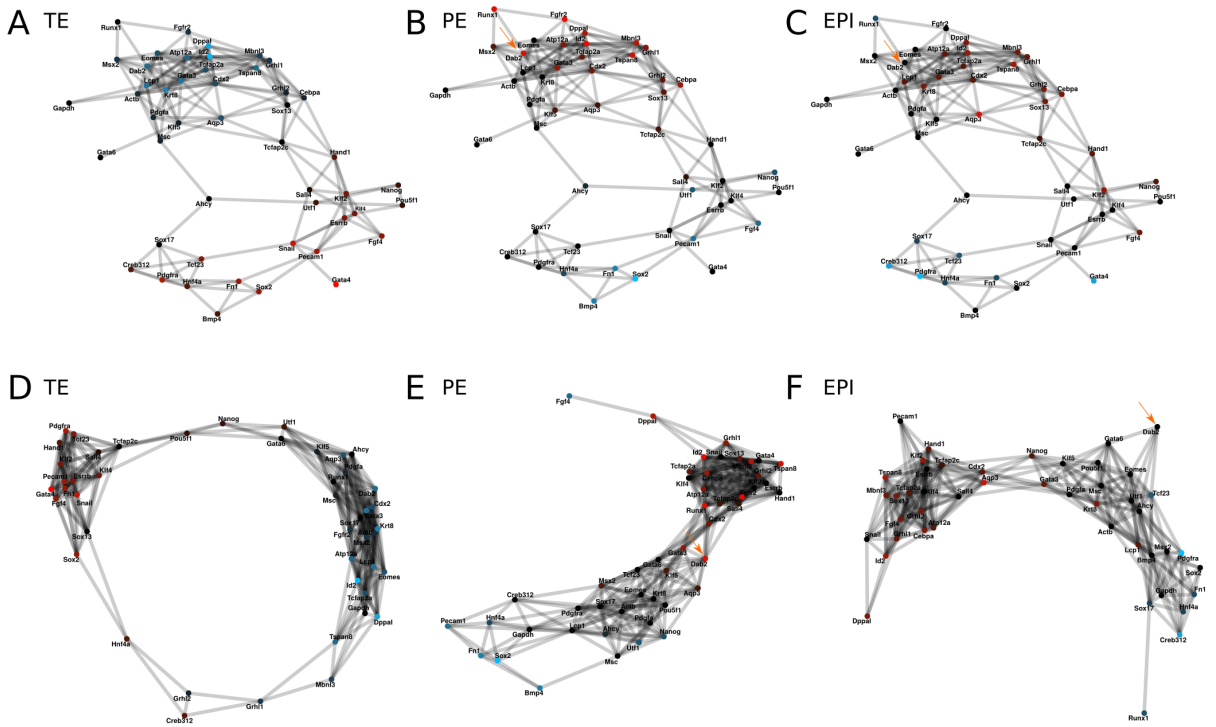

Figure S20: Zygote to Blastocyst GCNs (Guo 2010). We performed GCNs reconstructions using all cells such that we show differentially upregulated genes in shades of red and downregulated genes in shades of blue for **(A)** TE branch, **(B)** PE branch, **(C)** EPI branch. The same color schema was used for GCNs using trajectory specific cells for **(D)** TE trajectory, **(E)** PE trajectory, **(F)** EPI trajectory.

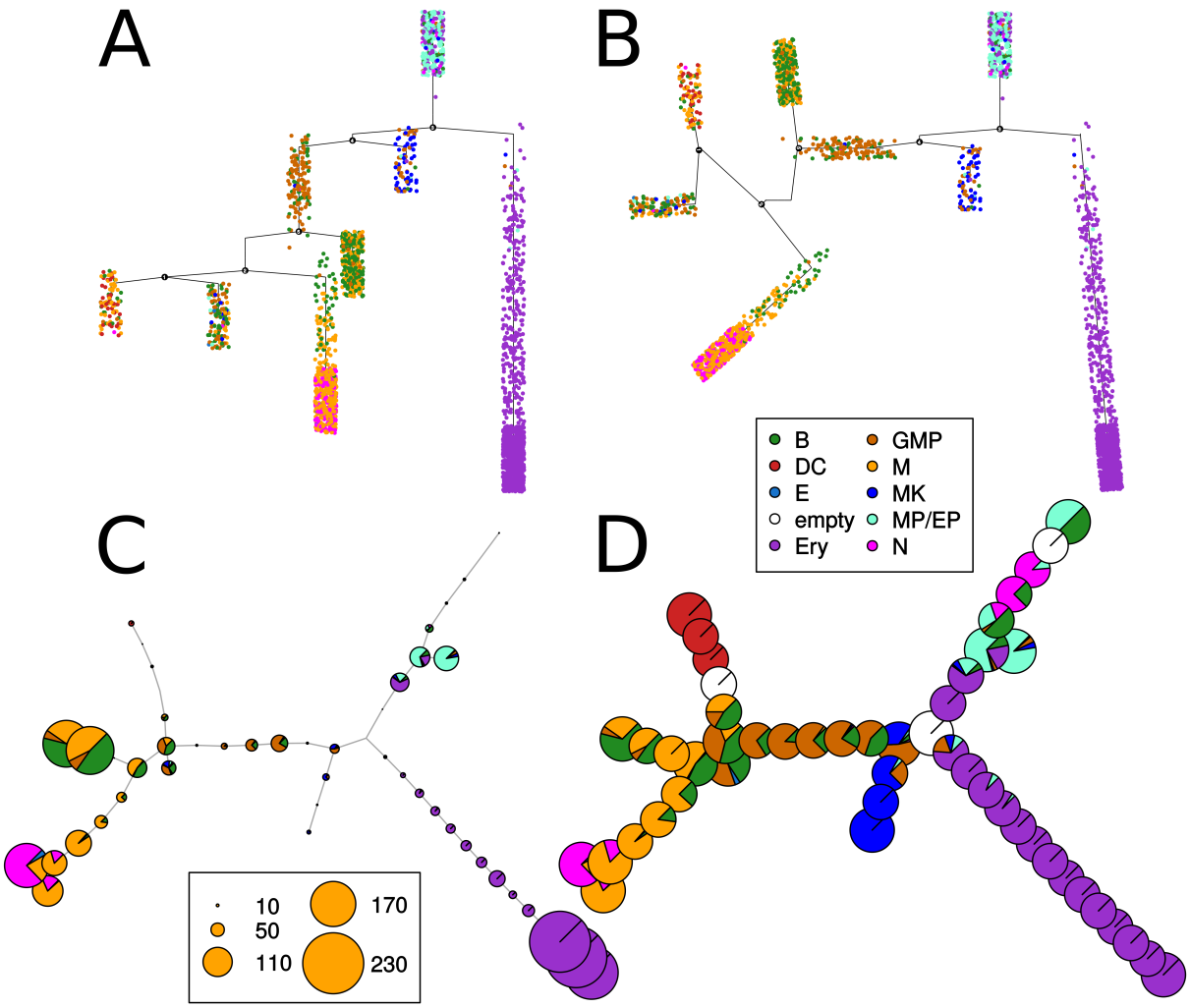

Figure S21: MERLoT improves upon Monocle2's reconstruction. **(A)** The flattened tree schematic provided by Monocle2. The erythrocyte branch (purple) is clearly separated, and while other branches contain similar progenitor groups, the separation is not optimal. **Abbreviations:** B: basophil (green); DC: dendritic cell (red); E: eosinophil (light blue); Ery: erythrocyte (purple); GMP: granulocyte and monocyte progenitor (brown); M: monocyte; MK: megakaryocyte; MP/EP: multipotent myeloid and erythroid progenitors; N: neutrophil (orange) **(B)** The same topology but arranged in slightly different manner to facilitate comparison with the MERLoT reconstructions. **(C)** The MERLoT reconstruction of the same tree (based on the same coordinates): The branches are more homogeneous. The colour composition of each pie chart corresponds to the types of cells assigned to that elastic tree node. The size of the pie charts is proportional to the number of cells assigned to each node. **(D)** The same reconstruction but without the number of cells.

References

[1] Saitou, N. and Nei, M. (1987) The neighbor-joining method: a new method for reconstructing phylogenetic trees.. *Molecular biology and evolution,* **4**(4), 406–425.

[2] Guo, G., Huss, M., Tong, G. Q., Wang, C., Sun, L. L., Clarke, N. D., and Robson, P. (2010) Resolution of cell fate decisions revealed by single-cell gene expression analysis from zygote to blastocyst. *Developmental cell,* **18**(4), 675–685.

[3] Paul, F., Arkin, Y., Giladi, A., Jaitin, D. A., Kenigsberg, E., Keren-Shaul, H., Winter, D., Lara-Astiaso, D., Gury, M., Weiner, A., et al. (2015) Transcriptional heterogeneity and lineage commitment in myeloid progenitors. *Cell,* **163**(7), 1663–1677.

[4] Grün, D., Kester, L., and Van Oudenaarden, A. (2014) Validation of noise models for single-cell transcriptomics. *Nature methods,* **11**(6), 637.

[5] Harris, K. D., Bengtsson Gonzales, C., Hochgerner, H., Skene, N. G., Magno, L., Katona, L., Somogyi, P., Kessaris, N., Linnarsson, S., and Hjerling-Leffler, J. (2017) Classes and continua of hippocampal CA1 inhibitory neurons revealed by single-cell transcriptomics. *bioRxiv,*.

[6] Papadopoulos, N., Parra, R. G., and Soeding, J. (2019) PROSSTT: Probabilistic simulation of single-cell RNA-seq data for complex differentiation processes.. *Bioinformatics,*.

[7] Vinh, N. X., Epps, J., and Bailey, J. (December, 2010) Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance. *J. Mach. Learn. Res.,* **11**, 2837–2854.

[8] Campello, R. J. G. B. and Hruschka, E. R. (2009) On comparing two sequences of numbers and its applications to clustering analysis. *Information Sciences,* **179**(8), 1025–1039.

[9] Mao, Q., Wang, L., Goodison, S., and Sun, Y. (2015) Dimensionality Reduction Via Graph Structure Learning. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* New York, NY, USA: ACM KDD '15 pp. 765–774.

[10] Street, K., Risso, D., Fletcher, R. B., Das, D., Ngai, J., Yosef, N., Purdom, E., and Dudoit, S. (Jun, 2018) Slingshot: cell lineage and pseudotime inference for single-cell transcriptomics. *BMC Genomics,* **19**(1), 477.

[11] Hastie, T. and Stuetzle, W. (1989) Principal Curves. *Journal of the American Statistical Association,* **84**(406), 502–516.

[12] Scrucca, L., Fop, M., Murphy, T. B., and Raftery, A. E. (2016) mclust 5: clustering, classification and density estimation using Gaussian finite mixture models. *The R Journal,* **8**(1), 205–233.

[13] Haghverdi, L., Buettner, F., and Theis, F. J. (2015) Diffusion maps for high-dimensional single-cell analysis of differentiation data. *Bioinformatics,* **31**(18), 2989–2998.

[14] Kruskal, W. H. and Wallis, W. A. (1952) Use of Ranks in One-Criterion Variance Analysis. *Journal of the American Statistical Association,* **47**(260), 583–621.

[15] Trapnell, C. (2015) Defining cell types and states with single-cell genomics. *Genome research,* **25**(10), 1491–1498.

[16] Qiu, X., Mao, Q., Tang, Y., Wang, L., Chawla, R., Pliner, H. A., and Trapnell, C. (2017) Reversed graph embedding resolves complex single-cell trajectories. *Nature methods,* **14**(10), 979–982.

3. Simulating complex single-cell trajectories with PROSSTT

3.1. Introduction

Following the rise of single-cell transcriptomics, a plethora of methods for trajectory inference have been developed, going from 10 in 2016 [23] to over 70 by the end of 2018 [142, 201]. With this explosion of available methods the need to quantify their performance became pressing. In the absence of real data with known developmental times this is challenging. The corresponding progress in experimental protocols (see Fig. 1.1) made it clear that soon complex data from differentiations with multiple cell fate decisions would be available (as it was; see for example [41, 18, 128]), and it was not clear if the available tools would be up to the task.

In this manuscript we describe software to simulate differentiations with branching structures of any complexity and scRNA-seq count data that could plausibly arise from such a process. PROSSTT (PRObabilistic Simulation of Single-cell Tree-like Topologies) simulates a small number of expression programs that change their expression through developmental time, and models genes as linear combinations of expression programs, achieving correlation in gene expression in an indirect manner. It uses negative binomial distributions to draw the count data, but the parametrization allows Poisson or quasi-Poisson sampling.

PROSSTT produces data with realistic summary statistics, and can learn its hyperparameters from real data. If a trajectory inference algorithm can represent a differentiation in an appropriate manner, PROSSTT can simulate realistic data from the underlying process.

Nikolaos Papadopoulos performed the research and programming. Gonzalo Parra consulted. Nikolaos Papadopoulos and Johannes Soeding jointly designed the research and wrote the manuscript.

In a collaboration effort with labs from all over the world, the group of Luca Pinello organized a platform for the comparison of trajectory inference algorithms, as part of the Human Cell Atlas infrastructure initiative [134]. PROSSTT simulations were included in the platform, and we took part in the discussions shaping the project.

Availability

PROSSTT is published in Oxford Bioinformatics (Oxford University Press, DOI 10.1093).

Bioinformatics, 2019, 1–3
doi: 10.1093/bioinformatics/btz078
Advance Access Publication Date: 1 February 2019
Applications Note

Single-cell transcriptomics

PROSSTT: probabilistic simulation of single-cell RNA-seq data for complex differentiation processes

Nikolaos Papadopoulos, Parra R. Gonzalo and Johannes Söding*

Quantitative and Computational Biology, Max Planck Institute for Biophysical Chemistry, Göttingen 37077, Germany

*To whom correspondence should be addressed.

Associate Editor: Jonathan Wren

Received on June 29, 2018; revised on December 21, 2018; editorial decision on January 25, 2019; accepted on January 31, 2019

Abstract

Summary: Cellular lineage trees can be derived from single-cell RNA sequencing snapshots of differentiating cells. Currently, only datasets with simple topologies are available. To test and further develop tools for lineage tree reconstruction, we need test datasets with known complex topologies. PROSSTT can simulate scRNA-seq datasets for differentiation processes with lineage trees of any desired complexity, noise level, noise model and size. PROSSTT also provides scripts to quantify the quality of predicted lineage trees.

Availability and implementation: https://github.com/soedinglab/prosstt.

Contact: soeding@mpibpc.mpg.de

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Recent advances in single-cell RNA sequencing (scRNA-seq) (Klein *et al.*, 2015; Macosko, 2015) make it possible to generate expression profiles for thousands of cells. Clustering the transcriptomic snapshot of a cell population reveals cell types (Trapnell, 2015), and ordering the cells according to their progress through differentiation reconstructs cellular lineage trees, offering insights into czomplex processes such as organogenesis (Camp *et al.*, 2017). The change in gene expression along the reconstructed trees gives us unprecedented, time-resolved data to quantitatively investigate the gene regulatory processes underlying cellular development.

As more and more complex processes are investigated, there will be a need to derive lineage trees of topologies more complex than linear or singly-branched ones. Also, with various methods already published (Rostom *et al.*, 2017) and more being developed, the need to quantify method performance is becoming more pressing. With the available data, assessing method performance is challenging as there are no datasets with known ground truth, i.e. data with known intrinsic developmental time and cell identity. These needs can be addressed by simulating realistic scRNA-seq datasets of complex dynamic processes.

Tools like Splatter (Zappia *et al.*, 2017) and dyngen (Saelens *et al.*, 2018) can simulate scRNA-seq data from lineage trees, however both have limitations. In particular, Splatter does not explicitly model coordinated change in gene expression, which results in tree segments that are in truth non-adjacent being placed close to each other. This happens in gene expression space as well as after dimensionality reduction (Supplementary Section S5). Additionally, Splatter doesn't provide a global pseudotime for the simulated cells, reducing its usefulness in the context of the evaluation of tree inference methods. Dyngen is built around a gene regulatory network that gives rise to a certain network topology. This requires users to design the regulatory network or use one of the pre-generated modules, which limits the complexity of the topologies that can be simulated.

Here we present PROSSTT (PRObabilistic Simulation of Single-cell RNA-seq Tree-like Topologies), a python package for simulating UMI counts from scRNA-seq experiements of complex differentiation pathways.

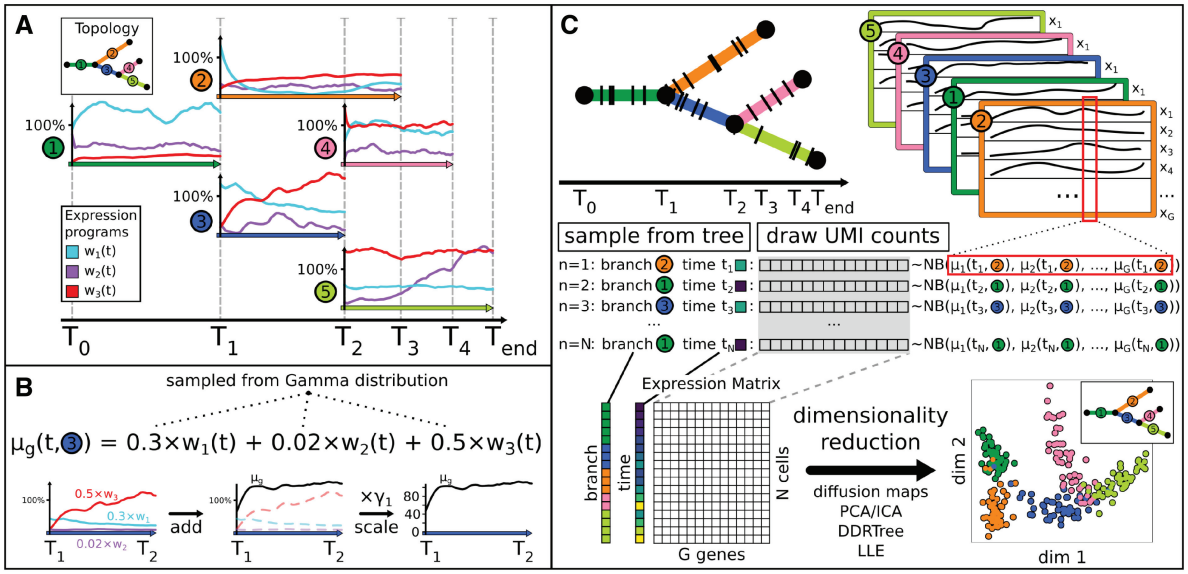

Fig. 1. PROSSTT models the single-cell RNA-seq transcriptomes of cells differentiating along a (user-supplied or sampled) lineage tree. (**A**) A small number of gene expression programs is simulated by random walk along each of the tree branches (number of steps = integer branch length). Here, a double bifurcation is regulated by three expression programs. (**B**) Relative expected gene expression $\mu_g(t,b)$ is computed as weighted sum of the expression programs with randomly sampled weights (here: gene *g* in branch 3). Expected expression values are obtained by multiplying with a gene-dependent sampled scaling factor. (**C**) Cells are sampled from the tree as pairs of pseudotime *t* and branch *b*. For each pair, the corresponding average gene expression is retrieved and UMI counts sampled using a negative binomial distribution. Low-dimensional representations of the resulting gene expression matrix are similar to those of real data (Supplementary Section S1) and capture the lineage tree topology [diffusion map created with destiny (Angerer *et al.*, 2016)]

2 Model

PROSSTT generates simulated scRNA-seq datasets in four steps

1. Generate tree: The topology of the lineage tree (number of branches, connectivity) and the length of each branch are read in or, alternatively, sampled. The integer branch lengths give the number of steps of the random walk (see next point) and correspond to the pseudotime duration [Fig. 1A (inset)]. The topology can also be linear.

2. Simulate average gene expression along tree: Gene expression levels are linear mixtures of a small number K (default: scales with number of bifurcations) functional expression programs w_k. For each tree segment, we simulate the time evolution of expression programs by random walks with momentum term (see Fig. 1A and Supplementary Material). The mean expression of gene g in tree branch b at pseudotime t is a weighted sum of the K different programs k: $\mu_g(t,b) = \sum_{k=1}^{K} w_k(t,b) h_{k,g}$ (Fig. 1B). The weights $h_{k,g}$ are drawn from a gamma distribution (Supplementary Section S2.2).

3. Sample cells from tree: We offer multiple ways of sampling cells from a lineage tree: (i) sampling cells homogeneously along the tree, (ii) sampling centered diffusely around selected tree points, (iii) sampling with user-supplied density and (iv) specifying the velocity with which the process progresses and sampling the resulting density. (Fig. 1C left, Supplementary Section S2.3).

4. Simulate UMI counts: We simulate unique molecular identifier (UMI) counts using a negative binomial distribution. First, a scaling factor s_n for the library size is drawn randomly for each cell n (see Supplementary Section S2.4). Following Grün *et al.* (2014) and Harris *et al.* (2018), we make the variance σ_g^2 depend on the expected expression $s_n\mu_g$ as $\sigma_{ng}^2 = \alpha_g(s_n\mu_g)^2 + \beta_g(s_n\mu_g)$. If $\mathbf{x}_n(t,b) = (x_1, x_2, \ldots, x_G)$ is a cell at pseudotime t and branch b, the transcript counts are $x_{ng}(t,b) \sim \mathrm{NegBin}\left(s_n\mu_g(t,b), \sigma_{ng}^2(t,b)\right)$ (Fig. 1C, right). For each of N cells and each of G genes we draw the number of UMIs from the negative binomial, resulting in an $N \times G$ expression matrix, which can serve as input for tree inference algorithms.

Users can specify the topology of the lineage tree (any connected acyclic graph is acceptable), assign branch pseudotime lengths, adjust parameters for the gene expression programs and control the noise levels in the data. Default parameter values for α_g, β_g and the base gene expression values were set in the range of parameters of real datasets (Supplementary Section S3). If provided with a real dataset, PROSSTT can learn hyperparameters that will generate simulated data with similar summary statistics.

3 Application

We generated 10 sets of 100 simulations each, for different degrees of topology complexity (from 1 up to 10 bifurcations). In another study, we used this dataset to assess the performance of our tool MERLoT and other methods (Parra *et al.*, 2018). We provide scripts with implementations of appropriate quality measures as well as the pipeline to generate the simulations and evaluate predictions by state-of-the-art software.

PROSSTT is capable of producing simulations with the summary statistics of true datasets, and can reproduce data faithfully in cases where the underlying lineage tree is available.

4 Conclusions

PROSSTT simulates scRNA-seq data for complex differentiation processes. Low-dimensional visualizations produced by tree reconstruction tools resemble those of real datasets. Increasingly complex datasets with uncertain biological ground truth are becoming available. PROSSTT can help the development of methods that can reconstruct such complex trees by facilitating their quantitative assessment. Furthermore, the modular nature of the software allows

for easy extensions, for example PROSSTT could serve to test the influence of noise models and give biological insights into how to model and interpret scRNA-seq data.

Funding

RGP is a long term EMBO postdoctoral fellow (ALTF 212-2016).

Conflict of Interest: none declared.

References

Angerer,P. *et al.* (2016) destiny: diffusion maps for large-scale single-cell data in R. *Bioinformatics*, **32**, 1241–1243.

Camp,J.G. *et al.* (2017) Multilineage communication regulates human liver bud development from pluripotency. *Nature*, https://www.nature.com/articles/nature22796.ris.

Grün,D. *et al.* (2014) Validation of noise models for single-cell transcriptomics. *Nat. Methods*, **11**, 637–640.

Harris,K.D. *et al.* (2018) Classes and continua of hippocampal CA1 inhibitory neurons revealed by single-cell transcriptomics. *PLoS Biol.*, **16**, 1–37.

Klein,A.M. *et al.* (2015) Droplet barcoding for single-cell transcriptomics applied to embryonic stem cells. *Cell*, **161**, 1187–1201.

Macosko,E.Z. *et al.* (2015) Highly parallel genome-wide expression profiling of individual cells using nanoliter droplets. *Cell*, **161**, 1202–1214.

Parra,R.G. *et al.* (2018) *Reconstructing Complex Lineage Trees from scRNA-seq Data using MERLoT.* Cold Spring Harbor Laboratory. doi: 10.1101/261768

Rostom,R. *et al.* (2017) Computational approaches for interpreting scRNA-seq data. *EBS Lett.*, **591**, 2213–2225.

Saelens,W. *et al.* (2018) *A Comparison of Single-cell Trajectory Inference Methods: Towards more Accurate and Robust Tools*. Cold Spring Harbor Laboratory, doi:10.1101/276907.

Trapnell,C. (2015) Defining cell types and states with single-cell genomics. *Genome Res.*, **25**, 1491–1498.

Zappia,L. *et al.* (2017) Splatter: simulation of single-cell RNA sequencing data. *Genome Biol.*, **18**, 174.

Supplementary Material for *PROSSTT: probabilistic simulation of single-cell RNA-seq data for complex differentiation processes*

Nikolaos Papadopoulos, R. Gonzalo Parra, Johannes Söding

Quantitative and Computational Biology Group, Max Planck Institute for Biophysical Chemistry, Göttingen, 37077, Germany

Detailed documentation for PROSSTT is available at `https://prosstt.readthedocs.io/en/latest/`. Multiple examples are available in the form of Jupyter notebooks in the `examples/` folder of the git repository: `https://github.com/soedinglab/prosstt/blob/master/examples`.

1 Diffusion maps of real and simulated datesets (Fig. S1)

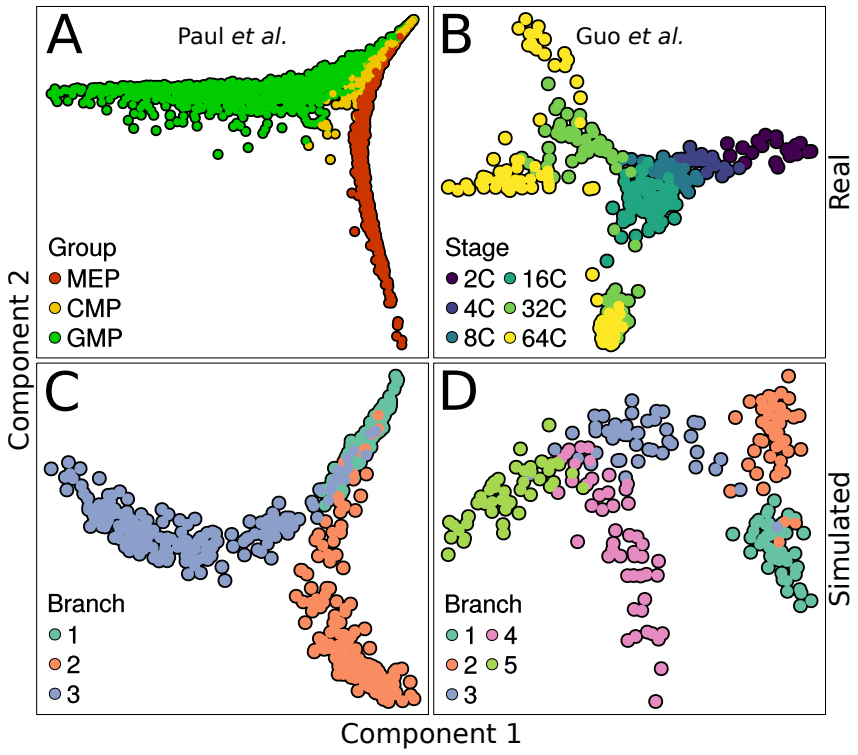

Figure S1: Diffusion maps of real (left) and simulated (right) datasets.

A: Mouse bone marrow cells were profiled using scRNA-seq (Paul *et al.*, 2015). An expression matrix containing 2730 cells and 3459 genes was obtained, and the cells were grouped in 19 transcriptionally homogenous clusters, as described by the authors. Using the expression of known marker genes, the cells were assigned to three broad clusters - megakaryocyte/erythroid progenitors, common myeloid progenitors, and granulocyte/macrophage progenitors. A diffusion map with default parameters was calculated with *destiny*, and the first two diffusion components were used to plot the figure.

B: Mice embryos up to and including the 64 cell stage were profiled using qRT-PCR (Guo *et al.*, 2010). An expression matrix containing 428 cells and 48 genes was obtained. The embryonal stage of each cell is known. The protocol described in the *destiny* vignette for data normalization was used in order to derive the final gene expression matrix. A diffusion map with global *sigma* option was calculated with *destiny*. To simplify visualization, the third diffusion component was rotated by 30 degrees onto the first. The transformed 1st-3rd diffusion component and the 2nd were used to plot the figure.

C: scRNA-seq data of a differentiation process with a single bifurcation, simulated using PROSSTT. This particular simulated data belongs to the simulations produced for the benchmark of the tool "MERLoT" (single bifurcation, simulation #38, Parra *et al.* (2018)). It contains 758 genes, simulated using 10 functional components. Each branch was 50 pseudotime

units long. The variance parameters for the negative binomials were sampled from $\alpha_g \sim e^x, x \in N(\log(0.2), 0.3)$ and $\beta_g \sim e^x, x \in N(\log(2), 0.3)$ respectively. The simulation for MERLoT originally contained 150 cells, but to improve visualization, the average gene expression trajectories from simulation #38 (simulation `benchmark1/test38/` in `http://wwwuser.gwdg.de/~compbiol/merlot/`) were reused to sample the whole lineage tree 4-fold, resulting in 600 cells. The first and second diffusion components were used for visualization.

D: scRNA-seq data of a differentiation process with a double bifurcation, simulated using PROSSTT. This particular simulated data belongs to the simulations produced for the benchmark of the tool "MERLoT" (double bifurcation, simulation `benchmark1/test24/` in `http://wwwuser.gwdg.de/~compbiol/merlot/`). It contains 250 cells and 836 genes, simulated using 10 functional components. Each branch is 50 pseudotime units long, and their connectivity (lineage tree topology) is described by $[[0, 1], [0, 2], [2, 3], [2, 4]]$, where branch 0 is the branch with pseudotime 0, where the differentiation starts. The variance parameters for the negative binomials were sampled from $\alpha_g \sim e^x, x \in N(\log(0.2), 0.3)$ and $\beta_g \sim e^x, x \in N(\log(2), 0.3)$ respectively.

The R script as well as the data that were used to produce the figures can be found in the `https://github.com/soedinglab/prosstt-scripts/` repository.

2 PROSSTT simulation structure

A typical PROSSTT simulation is performed in the following steps, which will be explained in detail below.

0. Initialize tree:
   - 0a. Set number of expression programs K.
   - 0b. Set number of genes G.
1. Generate tree:
   - 1a. Set lineage tree topology.
   - 1b. Set length of each branch in pseudotime units.
2. Simulate average gene expression along tree.
   - 2a. Simulate expression programs for each branch.
   - 2b. Randomly assign programs to genes.
   - 2c. Calculate relative average gene expression change.
   - 2d. Simulate base gene expression values.
   - 2e. Scale relative average gene expression change.
3. Sample cells from lineage tree according to a sampling strategy.
4. Sample variance parameters and simulate UMI counts for each gene g and cell n.

2.1 Generate tree

The various options available to initialize a tree are documented in `https://prosstt.readthedocs.io/en/latest/step1.html`. PROSSTT can simulate topologies in the form of connected acyclic graphs (such as linear differentiations or binary trees). Additionally, PROSSTT allows the simulation of committed lineages that separate later - two branches that will only diverge away from each other towards the end of their shared length.

2.2 Simulate average gene expression along the tree

2.2.a Simulate expression programs for each branch

In PROSSTT, gene expression is controlled by a small number of expression programs $K \ll G$. An expression program k encodes impulses for change in relative mean expression during pseudotime on a linear segment of the lineage tree (branch b). Intuitively, expression programs can be described as coordinated fold-change in average gene expression caused by any cellular process that has a time component.

For example, activation of a group of transcription factors A will upregulate some genes (expression increase over time) while transcription factor group B will cause downregulation (expression decrease over time) of some of the same genes. The influence of each factor is one expression program ("steady expression for 5 pseudotime units (PUs) and upregulation by 50% over 10 PUs", "downregulation by 30% over 15 PUs"). The fold-change in the expression of each gene will be an interplay

of the different influences (for a gene that is equally influenced by A and B it would be "downregulation by 30% over 5 PUs and upregulation by 20% over 10 PUs").

We simulate this change in every segment of the lineage tree with a random walk with a momentum term η, an amortized diffusion process that starts in $[0, 1.5]$ and moves freely. This diffusion takes place in log-space and is presented in Algorithm 1. For each program k in each branch b, $W_{b,k}[t]$ describes the location of the diffusing particle in time t, while $V_{b,k}[t]$ describes its velocity. The position of the particle at the time step $t+1$ is determined by its current position and its current velocity. In the following, let T_b be the length of the branch in pseudotime units, $\mathcal{N}(\mu, \sigma^2)$ a normal distribution with mean μ and variance σ^2 and $\mathcal{U}(0, 1.5)$ the uniform distribution between 0 and 1.5.

$W_{b,k}[0] = 0$;
$V_{b,k}[0] \sim \mathcal{N}(0, 0.2)$;
$\eta \sim \mathcal{U}(0, 1.5)$;
while $t \leq T_b$ **do**
  $W_{b,k}[t+1] \leftarrow W_{b,k}[t] + V_{b,k}[t]$;
  $\epsilon_t \sim N(0, 2/T_b)$;
  $V_{b,k}[t+1] \leftarrow 0.95V_{b,k}[t] + \epsilon_t + \eta V_{b,k}[t]$;
  $t \leftarrow t + 1$
end

Algorithm 1: Amortized diffusion in log-space

In each new time step $t+1$ the location $W_{b,k}[t+1]$ is updated by moving away from point $W_{b,k}[t]$ with speed $V_{b,k}[t]$. Then, the speed at the next step is updated: It is going to be almost the speed from the previous step ($0.95V_{b,k}[t]$) plus a small random change ϵ.

The overall change in position $\sum_{t=1}^{T_b} \epsilon_t$ is a sum of independent, identically distributed changes $\epsilon_t \sim \mathcal{N}(0, 1/T_b)$, and so is itself normally distributed with mean 0 and variance equal to the sum of the ϵ_t's.

$$\sigma^2_{total} = \sum_{t=1}^{T_b} \sigma^2_{\epsilon_t} = T_b \sigma^2_{\epsilon_t} = T_b \frac{1}{T_b} = 1 \tag{1}$$

This means that the expression fold change inside a linear segment is independent of the branch length in pseudotime units.

The diffusions of all K programs for branch b of the lineage tree can be summarized in a matrix $\mathbf{W}_b \in K \times T_b$, where each row is the random walk that represents program k in branch b as a function of pseudotime T_b ($W_{b,k}$). This process is repeated for all tree branches.

The number K of expression programs depends on the complexity of the simulated process. This is intuitively clear: one expression program would be enough to cause a single bifurcation, e.g. by upregulating all genes it influences in branch A and downregulating them in branch B. For two bifurcations, one expression program is not enough: if we apply the same "trick" at the end of branch A in order to create branches C (further upregulation) and D (downregulation), then the cells in branch D will be indistinguishable from the cells in branch A. A more nuanced approach is needed. This is further complicated by the fact that expression programs are random walks, and as such one cannot guarantee that each and every one of them will behave in a sufficiently different manner in parallel branches. PROSSTT therefore scales the number of expression programs quasi proprotionally to the number of branch points P present in the topology:

$$K = 5 \cdot P + x,\ x \sim \tilde{\mathcal{U}}(3, 20) \tag{2}$$

where $\tilde{\mathcal{U}}(3, 20)$ is the discrete uniform distribution between 3 and 20.

To alleviate the situation a bit more, PROSSTT imposes a correlation restriction on expression programs. Programs in the same branch are not allowed to have a Pearson's correlation coefficient above a certain threshold (default value 0.2). Due to the absence of redundancy, a small number of expression programs and their combinations are enough to simulate all the transcriptional variability.

2.2.b Randomly assign genes to programs

The relative average expression of gene g in tree branch b at pseudotime t is, by default, a weighted sum of the K different programs: $\mu_g(t, b) = \sum_{k=1}^{K} w_k(t, b) h_{k,g}$, where $h_{k,g} \sim \Gamma(\alpha)$. The coefficients can be summed up in a matrix $\mathbf{H} \in K \times G$. The shape α is set to the minimum of $1/K$ and 0.05. This reflects the fact that the number of expression programs grows with the topology complexity. If α had a static value, then simulations with more expression programs genes would have a higher average relative expression, leading to uneven simulations.

Alternatively, expression program contribution can also be set by choosing two expression programs to influence each gene. Here the coefficients are sampled from a beta distribution $B(\alpha, \beta)$ (default values: $\alpha = 2, \beta = 2$), while the contribution of all other expression programs is set to zero. Obviously, no additional balancing is required in this case.

2.2.c Calculate relative average gene expression

Multiplication of the coefficient matrix to the expression programs creates matrices that contain relative gene expression over pseudotime: $\mathbf{W}_b^T \times \mathbf{H} = \mathbf{M'}_b$ where $\mathbf{M'}_b \in T_b \times G$.

If expression programs on parallel branches (branches that are children of the same parent branch) correlate too much, then average gene expression will be very similar on both branches, with the result that they will not be easily distinguished. PROSSTT can resample the expression programs for the parallel branches until a certain proportion of genes is anticorrelated (Pearson correlation coefficient below 0). This is turned off by default.

Occasionally, some genes will have an extreme fold change, over a particular threshold (default: $e^8 \approx 3000$). If this happens, PROSSTT will re-simulate the offending branches to ensure that the expression of one gene does not dominate the dimensionality reduction of the simulated dataset.

2.2.d Simulate base gene expression

In order to translate fold-change in gene expression to change in absolute average gene expression over pseudotime, the relative expression encoded in $\mathbf{M'}_b$ must be scaled appropriately. This is done by sampling a base expression value for each gene. Intuitively, this is the expression of gene g in an unperturbed system; in the course of differentiation, differential expression of gene g happens respective to this base expression value.

Base expression values are sampled as $e^x, x \sim \mathcal{N}(0.8, 1)$. If a gene has an extreme fold-expression change over the lineage tree, then PROSSTT will try to pick a base expression value such that the absolute mean expression of the gene does not exceed a certain threshold value (default: 5000).

2.2.e Scale relative average gene expression change

This is a multiplication of the base expression values for each gene (vector $\hat{\mathbf{b}}$ of size G) with the relative expression matrix for each branch: $\mathbf{M}_b = \mathbf{M'}_b \cdot \hat{\mathbf{b}}$, where $\mathbf{M}_b \in T_b \times G$ is a matrix that contains absolute gene expression over time for branch b.

2.3 Sampling strategies

"Sampling" is the task of picking cells at the different pseudotime points along the lineage tree structure to simulate their expression. Formally, it consists of picking combinations of pseudotime points and branches (t, b). PROSSTT includes three sampling strategies. The modular nature of the program facilitates the inclusion of user-customized sampling strategies. Jupyter notebooks with example code for all cases are provided at `https://github.com/soedinglab/prosstt/tree/master/examples`.

2.3.a Sampling the whole lineage tree

Every point on the lineage tree is sampled n times. This mode produces simulations that are the easiest for trajectory inference algorithms, since it covers all possible instances of the model homogeneously. A variant of this mode will sample all pseudotime points n times, but will pick a branch randomly if multiple branches are available at the given pseudotime point. This mode still produces relatively easy simulations, with only a few small gaps that the algorithms must impute.

2.3.b Sampling a pseudotime series experiment

This option creates a simulation that resembles real data the most. scRNA-seq is usually performed as a time series experiment, where groups of cells are taken out of a culture or tissue at different time points $t'_1, t'_2, ..., t'_s$. In all snapshots, cells at the same sample time t'_i are going to be distributed around the same underlying differentiation time. In our model, given the amounts of cells sampled at each sample pseudotime point t'_i, we simulate cells with pseudotime values distributed around it. If multiple branches are possible, one is assigned randomly. This strategy produces the hardest simulations, as the gaps the algorithms must fill are going to be larger, and the density of cells along pseudotime is not uniform. For an example, refer to `https://github.com/soedinglab/prosstt/blob/master/examples/sample_pseudotime_series.ipynb`.

2.3.c Density sampling

Users might want to specify a sampling density to mimic relative cell abundance at different timepoints (e.g. a differentiation where most cells have already reached a cell fate but some linger in previous stages). Unless otherwise specified, the density of a simulated tree is uniform, i.e. all tree points are equally likely to be sampled. Consider `https://github.com/soedinglab/prosstt/blob/master/examples/density_sampling.ipynb` for an example of density sampling.

2.3.d Cell velocity

A different reason why parts of the lineage tree would be more or less dense is the speed of differentiation - cells might need a long time to make a cell fate decision (bottleneck) but differentiate quickly once that point has been passed. Users can specify differentiation velocities, which are translated to density behind the scenes. For a demonstration, refer to `https://github.com/soedinglab/prosstt/blob/master/examples/velocity.ipynb`.

Density sampling and pseudotime series can be combined to produce datasets with more realistic cell abundances. For an example, see `https://github.com/soedinglab/prosstt/blob/master/examples/combined_sampling.ipynb`.

2.4 Sampling variance hyperparameters and simulating UMI counts

We simulate unique molecular identifier (UMI) counts using a negative binomial distribution. First, a scaling factor s_n for the library size is drawn randomly for each cell n. The variance σ_g^2 depends on the average gene expression $s_n\mu_g$ as $\sigma_{ng}^2 = \alpha_g(s_n\mu_g)^2 + \beta_g(s_n\mu_g)$. For every cell n, sampled in time t and branch b, and each gene g, the transcript counts are $x_{ng}(t,b) \sim \text{NegBin}\left(s_n\mu_g(t,b), \sigma_{ng}^2(t,b)\right)$.

The hyperparameters are drawn from exponentiated normal distributions. Specifically:

- α_g is drawn from $e^x, x \in \mathcal{N}(\log(0.4), \log(2))$
- β_g is drawn from $e^x, x \in \mathcal{N}(\log(2), \log(2))$
- s_n is drawn from $e^x, x \in \mathcal{N}(\log(1), \log(0.7))$

3 Modelling count data from UMI protocols: motivating model and parameter choices

Grün *et al.* (2014) proposed that the negative binomial distribution, a discrete probability distribution that can model variance independently of the mean, provides a robust model for transcript count data, and supported their statement with analysis of a UMI dataset. We validated this proposition by fitting log-normal, Poisson, and negative binomial distributions on transcript count data from four recent publications (Grün *et al.*, 2014; Hashimshony *et al.*, 2016; Islam *et al.*, 2014; Zeisel *et al.*, 2015). In the first three a population of cells of the same type are analyzed, while Zeisel *et al.* (2015) find 9 major and 47 minor subtypes of cells in the mouse cortex and hippocampus. Cells of the same type (or subtype) can reasonably be expected to have similar expression values, and therefore we can use them to analyze various properties of the data. This included a total of 139966 genes in 3119 cells and 53 subgroups. It is important that no differentiating cells are included in these datasets, as average gene expression changes during differentiation, considerably complicating analysis of count data distribution. A detailed overview of these datasets can be found in Table S1.

The transcript count data was normalized per cell according to library size before fitting:

$$x_{ng} = \frac{x_{ng}}{S_n} \frac{\sum_{n=1}^{N} S_n}{N} \tag{3}$$

where $S_n = \sum_{g=1}^{G} x_{ng}$.

For each gene in each of the 53 subgroups, we calculated the Bayes Information Criterion for the negative binomial, Poisson and log-normal fits, according to eq. 4.139 in Bishop (2006). When considering the genes where one of the three models has positive or strong evidence in its favor (BIC difference of 2 or higher), the negative binomial distributions explain the transcript count distribution of a larger number of genes in most subgroups (see `prosstt-examples/plots.ipynb` notebook).

We used the log-normal implementation of the python *scipy* package (`scipy.stats.lognorm`). For the Poisson and negative binomial distributions, we implemented probability density functions and their derivatives and fitted them using maximum likelihood and the python `scipy.optimize` package with its implementation of the L-BFGS-B algorithm (Zhu *et al.*, 1997). For the negative binomial distributions we used the p, r parametrization and a formulation that can deal with fractional counts (after correcting for library size):

$$\Pr(X = k) = \frac{\Gamma(r+k)}{\Gamma(k+1)\,\Gamma(r)} p^k (1-p)^r \quad \text{for } k \in \mathbb{R} \tag{4}$$

In general, for PROSSTT, we assume a variance model where the variance of the expression of a gene depends on the square of the mean in the form $\sigma_g^2 = \alpha_g\mu_g^2 + \beta_g\mu_g$. This is the simplest negative binomial model that accounts for the overdispersion observed by Grün and others, while still allowing genes to have a Poisson expression pattern ($\alpha_g \simeq 0$ and $\beta_g \simeq 1$).

In order to fit α, β values from the data we need a dataset where the average gene expression values differ between groups of cells. This condition was only satisfied for the Zeisel dataset, which was therefore used. Fitting negative binomials generated parameters $p_{k,g}, r_{k,g}$ for each gene g in each cell cluster k. We used these values to calculate the mean $\mu_{k,g}$ and

variance $\sigma^2_{k,g}$ of the count distribution of each gene in each group. We then minimized the Euclidean distance between the calculated and postulated variance over all K clusters of cells:

$$f(\alpha_g, \beta_g | \mu_{k,g}, \sigma_{k,g}) = \sqrt{\sum_{k=1}^{K} \sigma^2_{k,g} - (\alpha_g \mu^2_{k,g} + \beta_g \mu_{k,g})} \tag{5}$$

The partial derivatives are (leaving out the g index and the data):

$$\frac{\partial f}{\partial \alpha} = \frac{-\sum_{k=1}^{K} \mu_k^2 (\sigma_k^2 - \alpha \mu_k^2 - \beta \mu_k)}{f(\alpha, \beta)} \tag{6}$$

$$\frac{\partial f}{\partial \beta} = \frac{-\sum_{k=1}^{K} \mu_k (\sigma_k^2 - \alpha \mu_k^2 - \beta \mu_k)}{f(\alpha, \beta)} \tag{7}$$

We conclude that it is appropriate to model the distribution of transcript count data for gene g with:

$$\mathbf{x}_g \sim \text{NegBin}(\mu_g, \sigma_g^2 = \alpha_g \mu_g^2 + \beta \mu_g) \tag{8}$$

These fits can be used to study appropriate ranges and distributions for the relevant hyperparameters of the negative binomial distributions. We fit log-normal distributions for α_g, β_g, cell library size, and average gene expression. The results are summarized in Table S1.

parameter	data average	PROSSTT default value
α mean	0.4187	0.4
α variance	6.1448	2
β mean	1.8375	2
β variance	1.9401	2
libr. size mean	0.9034	1
libr. size variance	1.5577	0.7
avg. gene expr. mean	0.9309	0.8
avg. gene expr. variance	1.7747	1

Table S1: Average parameter values from the fits and correponding default values in PROSSTT.

The default parameters of PROSSTT are picked very close to the average of the fitted parameter values, with the exception of average gene expression. This is because of how PROSSTT models gene expression; the values listed as PROSSTT defaults here are the values for base gene expression. Before PROSSTT uses average gene expression, it is scaled by its fold-change, resulting in a much broader distribution.

Detailed analysis, plots, as well as the code can be found online, in `https://github.com/soedinglab/prosstt-scripts/`. A Jupyter notebook for each of the four datasets contains data processing, normalization and fitting. Another notebook (`plot.ipynb`) contains plots and summaries of the results.

The choice of variance hyperparameters is of critical importance, as high variance will obscure the underlying structure of the data and make reconstruction of the lineage tree impossible (for example `https://github.com/soedinglab/prosstt/blob/master/examples/variance_sim.ipynb`).

4 Prediction evaluation and visualization

Lineage tree inference algorithms must succeed at two tasks: a) to place biologically similar cells close to each other and biologically dissimilar cells away from each other, and b) to place cells in such a way that their distance from the root of the lineage tree (in some measure) reflects their progress in the developmental pathway. The success of each task depends, to a degree, on the success of the other. To the best of our knowledge, there is no single measure that can encompass both at once, so we assess each separately.

4.1 Grouping similar cells

The first task can be interpreted in multiple ways. For example, since PROSSTT provides pseudotime labels for all cells, one can define two cells as "similar" if they are below a certain pseudotime threshold. Evaluation would consist of determining the nearest neighbours of each cell in the trajectory reconstruction and scoring the neighbourhood by its mean pseudotime distance from the cell. However, cell distances after dimensionality reduction depend on the difference in gene expression,

which is not proportional to the pseudotime. At branch points, for example, the average expression in one branch may change faster than on the other, leading to larger manifold distances between cells despite similar pseudotime distances.

Instead, one can use the lineage tree structure, and consider cells in the same segment (branch) sufficiently similar to each other. These can be compared to the branch assignments produced by a trajectory inference algorithm that reconstructs the lineage tree, reducing a) to a clustering problem. Given a cell c, let $Tr(c)$ be the cluster identity of c in the ground truth and $A(c)$ the cluster assigned to c by the algorithm. We define a pair of two cells c_i, c_j as a...

$$\begin{array}{ll} \text{true positive (TP)} & \text{if} \quad \mathrm{Tr}(c_i) = \mathrm{Tr}(c_j) \wedge \mathrm{A}(c_i) = \mathrm{A}(c_j) \\ \text{true negative (TN)} & \text{if} \quad \mathrm{Tr}(c_i) \neq \mathrm{Tr}(c_j) \wedge \mathrm{A}(c_i) \neq \mathrm{A}(c_j) \\ \text{false positive (FP)} & \text{if} \quad \mathrm{Tr}(c_i) \neq \mathrm{Tr}(c_j) \wedge \mathrm{A}(c_i) = \mathrm{A}(c_j) \\ \text{false negative (FN)} & \text{if} \quad \mathrm{Tr}(c_i) = \mathrm{Tr}(c_j) \wedge \mathrm{A}(c_i) \neq \mathrm{A}(c_j) \end{array}$$

Using these four values, many popular clustering indices can be computed: the F1 measure is $2PR/(P+R)$ and the Fowkles-Mallows index is $\sqrt{PR}$ (where P is the precision $TP/(TP+FP)$ and R is the recall $TP/(TP+FN)$). The Matthews Correlation Coefficient is

$$\frac{(TP \cdot TN - FP \cdot FN)}{\sqrt{((TP+FP)(TP+FN)(TN+FP)(TN+FN))}},$$

and the Jaccard Index is $TP/(TP+FP+FN)$, while the (unweighted) Rand index is $(TP+TN)/(N)$, where N is the number of all possible pairs ($N = TP+FP+TN+FN$).

While all the aforementioned indices and measures are well established, they are suboptimal performance indicators for the problem at hand, since they don't take cluster structure into consideration. As the simulated lineage trees become bigger and more complex, the number of cell pairs that are not in the same tree segment is going to grow much faster than the number of cell pairs in different tree segments. The number of TPs will grow much slower than the number of TNs, something that will, for example, inflate the the MCC index. Additionally, the number of possible FNs becomes much higher with each additional segment added to the tree, something that affects the Jaccard index and the F1 score. In short, these measures don't take into account the number of clusters in the data, and as such are suboptimal descriptors of clustering performance in comparison to NMI.

In our opinion the NMI is best for assessment of branch assignment, since it captures the amount of information present in the original clustering that was recovered by the prediction (values between 0 and 1). It corrects the effect of agreement solely due to chance between clusterings by using a hypergeometric background distribution and punishes overbranching and merging branches almost equally (Vinh *et al.*, 2010). Given the predicted and real cluster assignments U and V, NMI is defined as

$$\mathrm{NMI}(U,V) = \frac{\mathrm{MI}(U,V) - \mathbb{E}[\mathrm{MI}(U,V)]}{\max\{H(U),H(V)\} - \mathbb{E}[\mathrm{MI}(U,V)]}.$$

4.2 Evaluating cell ordering

Evaluating the performance of the different methods consists of quantifying the degree of agreement between the true/labeled and the predicted orderings provided by the different algorithms. Pseudotime only establishes a partial ordering on cells and no absolute time. Therefore, cells on branches not passed through one after the other cannot be compared. We find the longest path in the tree (from the root to a leaf) and compare the predicted pseudotime with the simulated one for the cells on this path. We used the Goodman-Kruskall index and the Kendall index (weighted and unweighted) to assess pseudotime prediction, as proposed by Campello and Hruschka (2009). All four indices count how many pairs of cells have been ordered correctly (S_+) or incorrectly (S_-) and produce similar results for the benchmark. The unweighted Goodman-Kruskall index is the simplest approach: $(S_+ - S_-)/(S_+ + S_-)$.

5 Divergence Analysis

We generated benchmark datasets for MERLoT (Parra *et al.*, 2018) with PROSSTT and Splatter (Zappia *et al.*, 2017). Multiple methods did not perform according to expectations. The issue was particularly obvious in the evaluation of pseudotime prediction, where Monocle2 (Trapnell *et al.*, 2014) sank to the level of random predictions, and MERLoT, which, even though it excelled at branch assignment, dropped off to quite low accuracy for large numbers of bifurcations. Additionally, TSCAN (Ji and Ji, 2016), which in the PROSSTT benchmark proved to be competent in branch assignment, returned completely nonsensical predictions for data simulated by Splatter.

As these methods have all been applied successfully on real data, we chose to examine the simulations. By visual inspection of the manifold embeddings we observed that the manifold embeddings of Splatter simulations often presented "short-circuits", where parts of the lineage tree seemed to fold back to preceding tree segments, effectively creating cycles.

While branches were often separated correctly (i.e. the cell clustering was correct), they were connected in wrong ways, decreasing the pseudotime prediction accuracy.

Since the manifold embeddings are a projection that aims to retain the most important dynamics in the data, we hypothesized that the reason for these short-circuits was that the offending branches did not diverge enough, or even converged towards previous tree segments. In terms of gene expression, this means that differentiating phenotypes (captured transcriptomes) were either not different enough or that they converged towards preceding phenotypes.

To test this hypothesis, we measured the distance of each waypoint (endpoint or branchpoint) from the origin, and normalized it by the average branch length in the path that led to it. The calculations described below were performed on the simulated gene expression data, after normalization for library size and log-transformation (see scripts `divergence_euclidean.R` and `divergence_diffmaps.R` in `https://github.com/soedinglab/merlot-scripts`).

As explained in Fig. S2, we retrieved the cells with minimum and maximum pseudotime in each branch (start cells and end cells respectively), calculated their pairwise distances, and defined their average d_b as the length of branch b. Next, we took the cells with globally minimum pseudotime and calculated their distances from all end cells. This is the minimum direct distance d'_b of each branch b from the origin. We normalized each d'_b with $\bar{d}_b = \frac{1}{|p|}\sum_{b' \in p} d_{b'}$, where p is the path $[b_0, b_1, ...b]$ from the origin branch b_0 to branch b. Effectively, this yields the distance from the origin measured in average branch lengths. Pooling the normalized distances from all paths of equal length (especially since all branches have the same length in the PROSSTT and Splatter simulations) shows how much the change in expression values correlates with pseudotime.

The divergence curve of PROSSTT (Fig. S3A) shows what we expected: monotonic growth (i.e. longer paths are on average further away from the origin) and values above 1, indicating that paths with two branches or more consistently end outside a one-branch-length radius from the origin, something that reduces the possibility of wrong assignments from the methods.

On the contrary, the divergence curve of Splatter (Fig. S3B) stays almost completely below 1 and even shows a slightly negative slope. This means that expression profiles of cells in later differentiation stages dont move further from the origin with increasing pseudotime length. We believe this happens because Splatter does not include co-regulation in its differentiation simulation model. This leads to the differences between branches being completely random, and while this may work to separate two diverging branches from each other, it does not seem to yield realistic diverging cell lineage trees.

As a control, we performed the divergence analysis in the diffusion maps created by destiny (Angerer *et al.*, 2016) for the benchmark (panels G,H in Fig. S3). Diffusion distance was proposed as a measure of cell similarity in the original paper Haghverdi *et al.* (2014), and was the most effective of the embeddings used in this study. We see the same trends as in gene expression space; the divergence curve of PROSSTT has positive slope and is consistently above 1, while the Splatter curve has a (clear) negative slope and stays below 1.

After quantifying divergence in both PROSSTT and Splatter simulations, we conclude that the simulations produced by Splatter have inherent characteristics that prevent algorithms that try to find a global structure (like MERLoT and monocle) from reaching their full potential. Consequently, we decided to only use simpler topologies for the Splatter benchmark (up to 4 bifurcations), where the impact of short-circuits was less dominant.

6 Learning from real data

While PROSSTT's default parameters are sampled from realistic ranges, users may want to simulate data with characteristics similar to real datasets, or even reproduce a real dataset. PROSSTT can learn hyperparameters and average gene expression from a real dataset, and produce simulations with similar summary statistics. This is demonstrated via Jupyter notebooks (in `https://github.com/soedinglab/prosstt/tree/master/examples` or the examples directory of the PROSSTT repository) for four different datasets, produced with different experimental protocols and from vastly different systems:

- Axolotl limb regeneration (10x Genomics) (Gerber *et al.*, 2018) `compare_axolotl.ipynb`
- Developing whole hydra (Drop-seq) (Siebert *et al.*, 2018) `compare_hydra.ipynb`
- Hematopoietic stem cells (index-omics) (Velten *et al.*, 2017) `compare_velten.ipynb`
- Multiple timepoints whole zebrafish embrya (Drop-seq) (Farrell *et al.*, 2018) `compare_zebrafish.ipynb`

In the case where a lineage tree has been reconstructed, PROSSTT can be configured to reproduce the process. We demonstrate an example (`https://github.com/soedinglab/prosstt/blob/master/examples/reproduce_axolotl.ipynb`) for axolotl limb regeneration where the lineage tree has been reconstructed with MERLoT (Parra *et al.*, 2018) on diffusion maps (Haghverdi *et al.*, 2014).

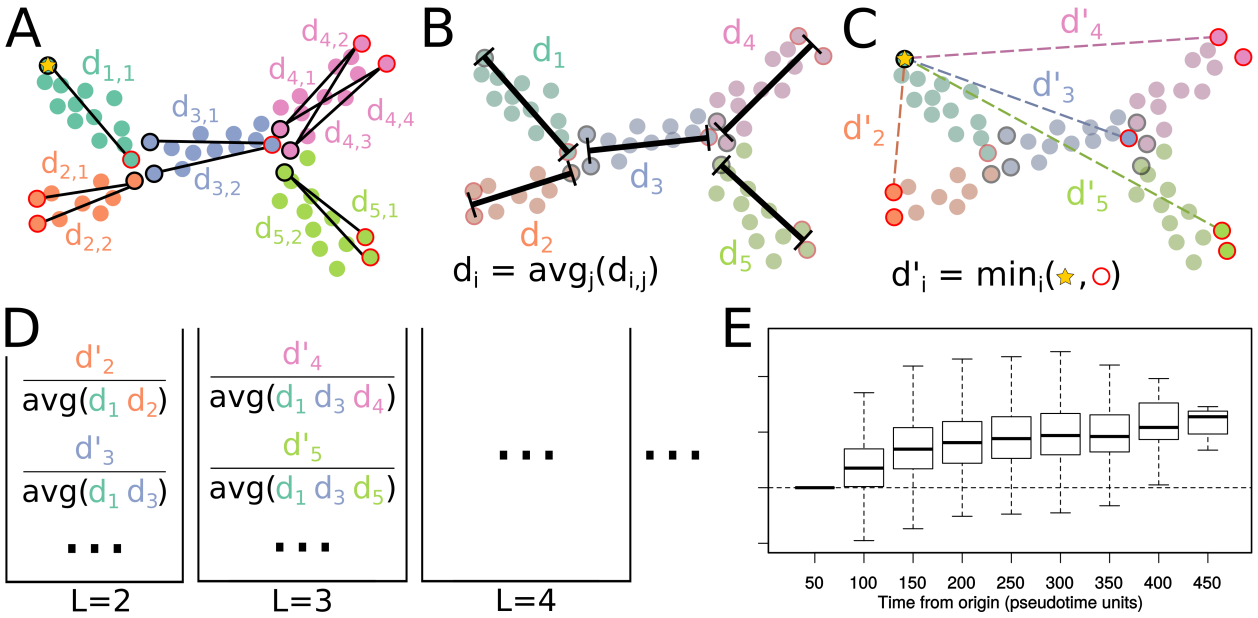

Figure S2: **Divergence analysis.** **(A)** We locate the cells with minimum and maximum pseudotime in each branch and calculate their pairwise distances. **(B)** The average of these distances is the branch length. **(C)** We calculate the minimum direct distance of the origin of the differentiation (minimum pseudotime of first branch) to the other waypoints (maximum pseudotime of each branch). **(D)** We take all on-tree paths from the origin to a waypoint and normalize the minimum direct distance via the average branch length in the path. After repeating steps (A)-(C) for all simulations in the set, we group these values by path length (in branches). **(E)** The boxplots of each path bin constitute the divergence curve of the simulations. If the branches all have the same pseudotime length, then distance to waypoints is equivalent to the pseudotime that has passed since the origin.

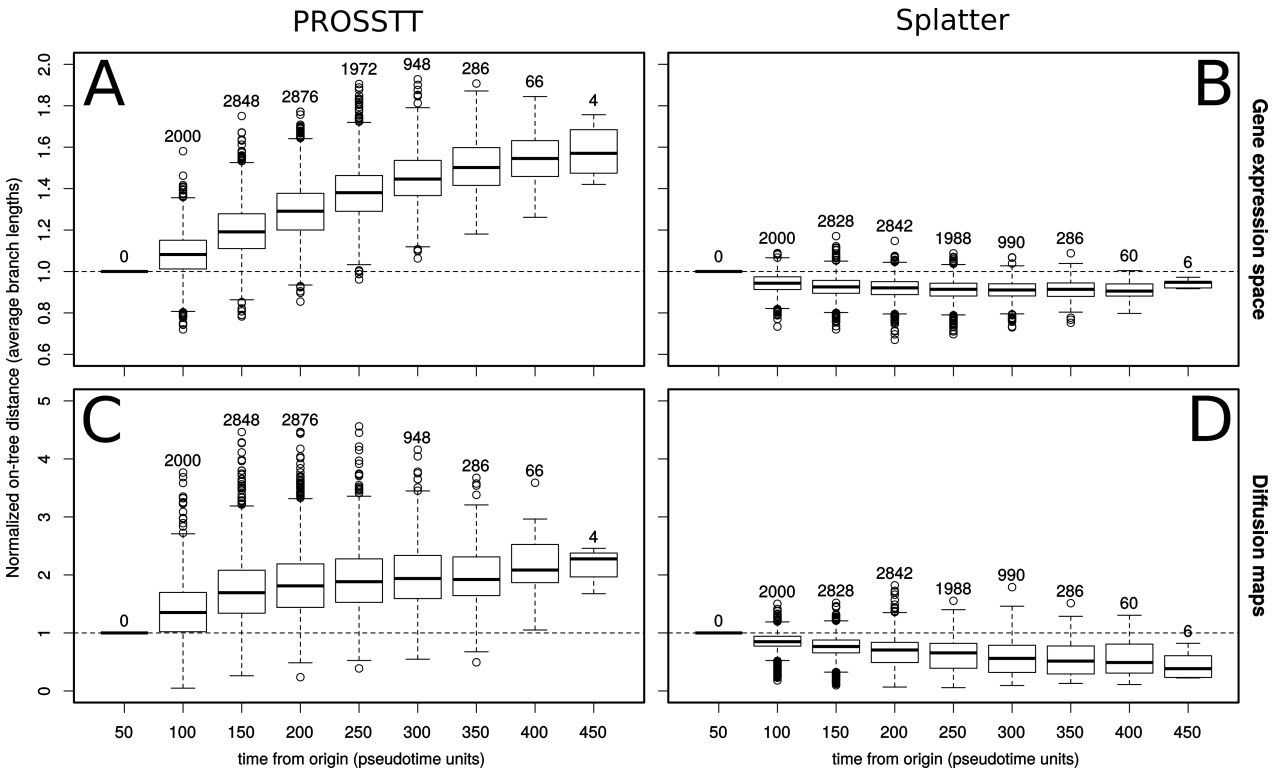

Figure S3: **Divergence analysis results.** Divergence analysis for PROSSTT (left) and Splatter (right) simulations in gene expression space (upper panels) or in diffusion space (lower panels).

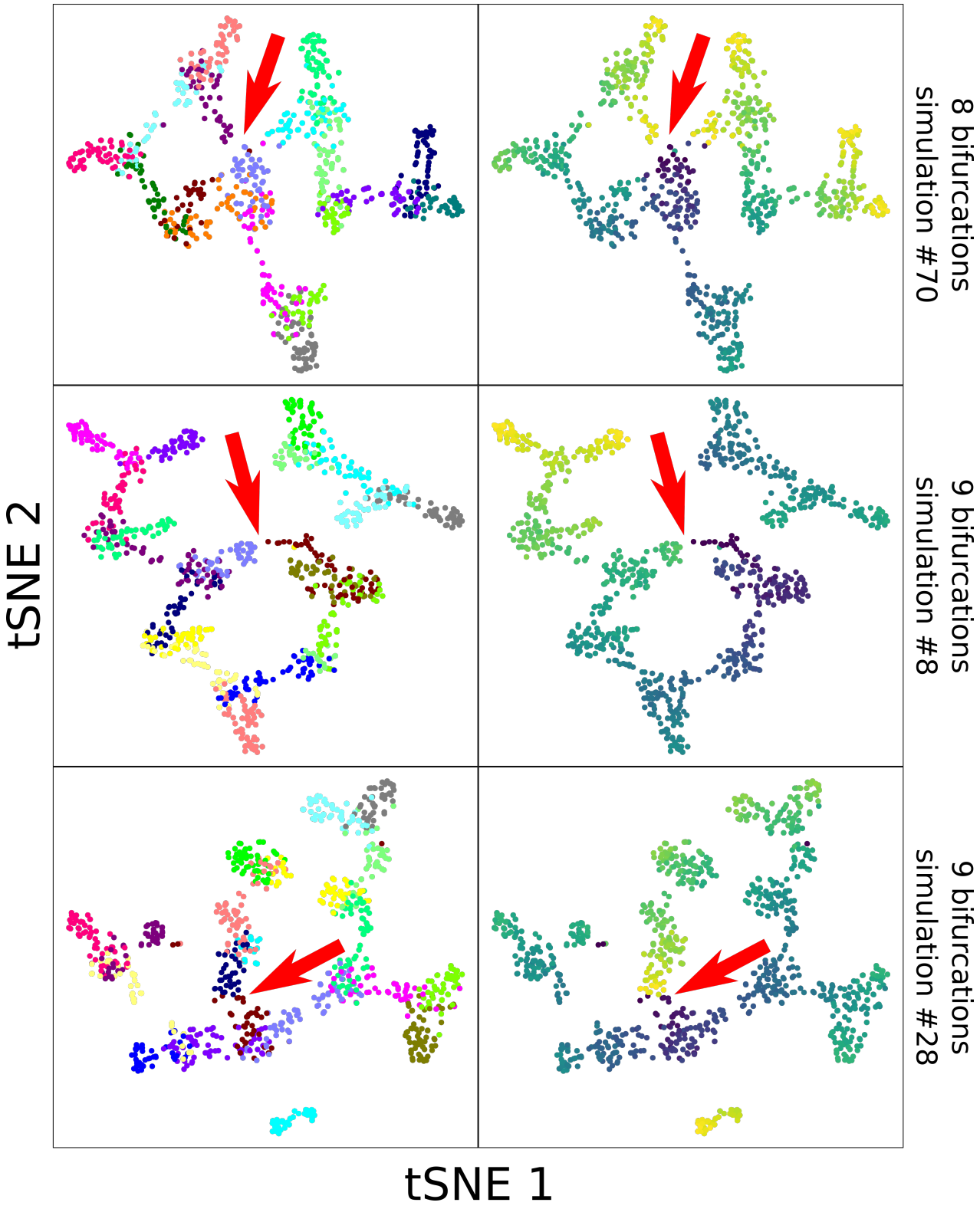

Figure S4: **Short circuits.** tSNE plots of Splatter simulations that show short-circuits. On the left side the colors denote the branches. On the right side the color denotes the pseudotime, from low (dark blue) to high (yellow). The red arrows point to the exact points where the short-circuit takes place. In all cases, cells with low and high pseudotime are so close to each other that the shape of the lineage tree is not clear, leading to problems in the tree reconstruction.

7 Programming and tools

Analysis and development was done in Python 3.6 (van Rossum, 1995) and R (R Core Team, 2017). IPython (Pérez and Granger, 2007) notebooks were used for the development of the simulation suite while Rstudio (RStudio Team, 2015) was used to write and analyze the evaluation code. PROSSTT was programmed in Python 3 using NumPy (van der Walt *et al.*, 2011) and SciPy (Jones *et al.*, 01). Plotting was performed with matplotlib (Hunter, 2007) and R base graphics. The `parallel` (Tange, 2011) bash utility was used to speed up simulations.

7.1 Implementation and availability

We provide code and scripts for the evaluation of trajectory inference predictions on data simulated by PROSSTT as the R package *prosstt*, available at `https://github.com/soedinglab/prosstt-r`. We have also implemented a plotting function that visualizes trajectory inference predictions on PROSSTT simulations.

Acknowledgements

The authors wish to think Tobias Gerber for providing the reconstructed tree object from the axolotl limb regeneration study (Gerber *et al.*, 2018).

References

Angerer, P., Haghverdi, L., Büttner, M., Theis, F. J., Marr, C., and Buettner, F. (2016). destiny: diffusion maps for large-scale single-cell data in r. *Bioinformatics*, **32**(8), 1241–1243.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Campello, R. J. G. B. and Hruschka, E. R. (2009). On comparing two sequences of numbers and its applications to clustering analysis. *Inf. Sci.*, **179**(8), 1025–1039.

Farrell, J. A., Wang, Y., Riesenfeld, S. J., Shekhar, K., Regev, A., and Schier, A. F. (2018). Single-cell reconstruction of developmental trajectories during zebrafish embryogenesis. *Science*, **3131**(April), eaar3131.

Gerber, T., Murawala, P., Knapp, D., Masselink, W., Schuez, M., Hermann, S., Gac-Santel, M., Nowoshilow, S., Kageyama, J., Khattak, S., *et al.* (2018). Single-cell analysis uncovers convergence of cell identities during axolotl limb regeneration. *Science*, page eaaq0681.

Grün, D. *et al.* (2014). Validation of noise models for single-cell transcriptomics. *Nature methods*, **11**(6), 637–640.

Guo, G. *et al.* (2010). Resolution of Cell Fate Decisions Revealed by Single-Cell Gene Expression Analysis from Zygote to Blastocyst. *Developmental Cell*, **18**(4), 675–685.

Haghverdi, L., Buettner, F., and Theis, F. J. (2014). Diffusion maps for high-dimensional single-cell analysis of differentiation data. *Bioinformatics*, **31**(18), 2989–2998.

Hashimshony, T. *et al.* (2016). CEL-Seq2: sensitive highly-multiplexed single-cell RNA-Seq. *Genome biology*, **17**(1), 77.

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, **9**(3), 90–95.

Islam, S. *et al.* (2014). Quantitative single-cell RNA-seq with unique molecular identifiers. *Nature methods*, **11**(1), 163–166.

Ji, Z. and Ji, H. (2016). Tscan: Pseudo-time reconstruction and evaluation in single-cell rna-seq analysis. *Nucleic acids research*, **44**(13), e117–e117.

Jones, E., Oliphant, T., Peterson, P., *et al.* (2001–). SciPy: Open source scientific tools for Python.

Parra, R. G., Papadopoulos, N., Ahumada-Arranz, L., El Kholtei, J., Treutlein, B., and Soeding, J. (2018). Reconstructing complex lineage trees from scRNA-seq data using MERLoT. *bioRxiv*.

Paul, F. *et al.* (2015). Transcriptional heterogeneity and lineage commitment in myeloid progenitors. *Cell*, **163**(7), 1663 – 1677.

Pérez, F. and Granger, B. E. (2007). IPython: a system for interactive scientific computing. *Computing in Science and Engineering*, **9**(3), 21–29.

R Core Team (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

RStudio Team (2015). *RStudio: Integrated Development Environment for R*. RStudio, Inc., Boston, MA.

Siebert, S., Farrell, J. A., Cazet, J. F., Abeykoon, Y., Primack, A. S., Schnitzler, C. E., and Juliano, C. E. (2018). Stem cell differentiation trajectories in hydra resolved at single-cell resolution. *bioRxiv*.

Tange, O. (2011). Gnu parallel - the command-line power tool. *The USENIX Magazine*, pages 42–47.

Trapnell, C., Cacchiarelli, D., Grimsby, J., Pokharel, P., Li, S., Morse, M., Lennon, N. J., Livak, K. J., Mikkelsen, T. S., and Rinn, J. L. (2014). Pseudo-temporal ordering of individual cells reveals dynamics and regulators of cell fate decisions. *Nature biotechnology*, **32**(4), 381.

van der Walt, S., Colbert, S. C., and Varoquaux, G. (2011). The numpy array: A structure for efficient numerical computation. *Computing in Science Engineering*, **13**(2), 22–30.

van Rossum, G. (1995). Python tutorial. Technical Report CS-R9526, Centrum voor Wiskunde en Informatica (CWI), Amsterdam.

Velten, L., Haas, S. F., Raffel, S., Blaszkiewicz, S., Islam, S., Hennig, B. P., Hirche, C., Lutz, C., Buss, E. C., Nowak, D., *et al.* (2017). Human haematopoietic stem cell lineage commitment is a continuous process. *Nature cell biology*, **19**(4), 271.

Vinh, N. X., Epps, J., and Bailey, J. (2010). Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *J. Mach. Learn. Res.*, **11**, 2837–2854.

Zappia, L., Phipson, B., and Oshlack, A. (2017). Splatter: simulation of single-cell RNA sequencing data. *Genome Biology*, **18**(1), 174.

Zeisel, A. *et al.* (2015). Cell types in the mouse cortex and hippocampus revealed by single-cell RNA-seq. *Science*, **347**(6226), 1138–42.

Zhu, C., Byrd, R. H., Lu, P., and Nocedal, J. (1997). Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Trans. Math. Softw.*, **23**(4), 550–560.

prosstt Documentation

Release 1.1.0

Nikolaos Papadopoulos

Jun 12, 2019

CONTENTS

CHAPTER

ONE

WHAT IS PROSSTT?

1.1 Introduction & Motivation

Single-cell RNA sequencing (scRNA-seq) is revolutionizing the study of molecular biology by making comprehensive transcriptomic profiles available for single cells. This is challenging the traditional definition and understanding of cell types, as scRNA-seq has demonstrated that transcriptional heterogeneity is present within cell populations previously thought homogeneous. Even more excitingly, scRNA-seq is providing unprecedented insights into cellular differentiation. Cellular lineage trees can be reconstructed from snapshots of single-cell transcriptomes of cells caught at various stages of their differentiation process. The study of gene expression change along the reconstructed trees can uncover the intricate regulatory processes that govern development and differentiation.

Dedicated tree inference algorithms are indispensable for the study of scRNA-seq differentiation data. While various algorithms have been developed and successfully used, there are two overlapping concerns going forward:

- while the experimental field is clearly trending towards bigger datasets with more complex lineage trees, most published data consists of tree topologies with one or at most two bifurcations. Hence most algorithms have never reconstructed complex lineage trees, and it is not clear how they would perform.
- even if such datasets were available, it is currently very difficult to quantify method performance, since there exist no datasets with known ground truths, i.e. data with known intrinsic developmental time and cell identities that can be compared to algorithm predictions.

PROSSTT addresses both needs by producing realistic scRNA-seq datasets of differentiating cells.

1.2 How does it work?

PROSSTT generates simulated scRNA-seq datasets in four steps:

1. **Generate tree:** the topology of the lineage tree and the length of each branch are either sampled or set by the user.
2. **Simulate average gene expression along tree:** gene expression levels are a linear mixture of a small number K (default K=10) of functional expression programs. The time evolution of each expression program is simulated by random walks with a momentum term in each tree segment. The contribution weight of each expression program is drawn randomly for each gene.
3. **Sample cells from tree:** Three different sampling strategies are available: (1) sampling every point on the tree, (2) sampling via a density function or (3) sampling diffusely around selected tree points.
4. **Simulate UMI counts:** We simulate unique molecular identifier (UMI) counts using a negative binomial distribution where the variance of gene expression depends on average gene expression.

The end result of a PROSSTT simulation is a counts matrix of N cells x G genes, as well as a vector that contains the branch assignments and pseudotime values for each simulated cell.

1.3 Target users

This tool should be ideal for bioinformaticians currently developing tree inference methods, or comparing multiple methods to each other. PROSSTT can create datasets of varying difficulty, complexity and size and help test the limits of a method. In time PROSSTT could be extended to produce simulations from the reconstructed lineage trees of real datasets or include alternative noise models. We hope that PROSSTT will, directly or indirectly, help give biological insights into how to model and interpret scRNA-seq data.

Additional information can be found in the paper and the Supplemental Material.

CHAPTER

TWO

INSTALLATION

2.1 Manual installation

PROSSTT was developed and tested in Python 3.5 and 3.6. For an optimal experience, we suggest Python 3.6. In order to run PROSSTT, the following Python libraries have to be installed:

- scipy (tested: 1.0.0)
- numpy (tested: 1.14)
- pandas (tested: 0.22.0)
- newick (tested: 0.8.0)

In order to reproduce the example notebooks and produce plots with *generate_simN.py* the following libraries are additionally required:

- matplotlib (tested: 2.0.0, 2.1.2)
- jupyter (tested: 1.0.0)
- scanpy (tested: 0.4.2+9.ged7130f)

After installing all libraries:

- download a PROSSTT release or clone the git repository.
- (optional) unpack the archive
- `cd prosstt/directory/`
- `pip install .`

2.2 Install via Bioconda

(work in progress)

CHAPTER

THREE

USING PROSSTT

Here we will go through the four steps of simulation creation in PROSSTT and explain the various alternatives available in each one.

Contents:

3.1 Generate tree

The aim of this step is to generate a `Tree` object. For illustration purposes, we will try to create the lineage tree for a differentiation with a single cell fate decision (single bifurcation) where one side of the differentiation lasts longer than the other. We want the simulated data to have a total of 1000 genes and be regulated by 15 expression programs:

```
          100
   70   ---B------
---A---|
        ---C--
          60
```

A single bifurcation, where branch A is connected to branches B and C. The pseudotime lengths of the branches are A:70, B:100 and C:60, respectively. No part of the lineage tree is populated more densely than others - that means that if we picked a cell from the tree in a random manner, all positions are equally probable.

This can happen in various ways:

3.1.1 Manual definition

The `Tree()` constructor can be called directly with a topology that describes branch connectivity, a pseudotime length for each branch, the total number of branches and branch points, as well as values for the number of genes and expression programs (modules) (see *Simulate average gene expression along tree*):

```
from prosstt.tree import Tree
t = Tree(topology=[["A", "B"], ["A", "C"]],
         time={"A":70, "B":100, "C":60},
         num_branches=3,
         branch_points=1,
         modules=15,
         G=1000,
         density=None,
         root="A")
```

If `density=None` is passed (or if no value for the parameter is given), PROSSTT will automatically calculate a uniform density for the lineage tree.

3.1.2 From a Newick-formatted string

A Newick-formatted string can be used instead; in this case PROSSTT will parse the topology from the Newick string. It is important to note that just like in the manual definition, the Newick tree describes branches and their connections:

```
import newick
from prosstt.tree import Tree

newick_string = "(B:100,C:60)A:70;"
newick_tree = newick.loads(newick_string)
t = Tree.from_newick(newick_tree, modules=15, genes=1000, density=None)
```

The `topology`, `time`, `num_branches`, `branch_points`, and `root` parameters can be parsed out of the newick tree. The same considerations about `density` as before apply.

3.1.3 Automatic tree generation

For higher order topologies (above 3 leaves in the tree) multiple topological arrangements are possible (also see Supplemental Material of the paper). As was the case in the MERLoT benchmark, sometimes users want to generate a tree topology with the number of bifurcations as the only input parameter. For example, for a lineage tree with 4 bifurcations (9 branches) where each branch has the same length, 50 pseudotime units:

```
from prosstt.tree import Tree
import string

branch_names = list(string.ascii_uppercase[:9])
time = {branch: 50 for branch in branch_names}
t = Tree.from_random_topology(branch_points=4, time=time, modules=15, genes=1000)
```

3.2 Simulate average gene expression along tree

Having obtained a `Tree` object (we will be calling it `tree`), the next step is then to simulate average gene expression along each branch. In PROSSTT, gene expression is controlled by a small number of gene expression programs. These can be imagined as instructions for gene expression; they describe change in relative gene expression (from 0% to 100% or even more).

3.2.1 Contribution coefficients

Each expression program can influence multiple genes, and the expression of each gene is the weighted sum of all expression programs. These contributions are controlled by coefficients sampled (by default) from a Gamma distribution:

```
coefficients = simulate_coefficients(tree, a=0.05)
```

If a simpler model is desired, the coefficients can also be sampled from a Beta distribution. In this case, each gene will be controlled by a maximum of two expression programs, with the contribution of all others set to zero:

```
coefficients = simulate_coefficients(tree, a=2, b=2)
```

3.2.2 Expression programs

Given the program-gene coefficients, we visit the branches of the tree breadth-first and simulate the expression programs. We can then calculate the relative average gene expression over pseudotime as the matrix product of the expression programs and the coefficient matrix. We adjust it so that the relative expression for each gene starts where it ended in the preceding branch:

```
from prosstt import simulation as sim
from prosstt import sim_utils as sut
import numpy as np

bfs = sut.breadth_first_branches(tree)

for branch in bfs:
    programs[branch] = sim.sim_expr_branch(tree.time[branch], tree.modules)
    rel_means[branch] = np.dot(programs[branch], coefficients)
    rel_means[branch] = sut.adjust_to_parent(rel_means, branch, topology)
```

Each expression program in each branch is a random walk with a momentum term that takes place in log space. PROSSTT requires that expression programs have a low correlation with each other (default: 0.2). This makes sure that a small number of (non-redundant) programs can encode all the variability needed.

At this point, PROSSTT enforces two quality checks: first, it requires that no expression program has a value above a cutoff (default is 8; with a relative expression value of `exp(8)`, a gene is upregulated to almost 3000 times its original expression level). Second, it requires that expression programs move differently in branches that are parallel in pseudotime (branches that share a branch point).

More specifically, PROSSTT calculates the Pearson correlation coefficient between the time course of each expression program in both branches. It then requires that a proportion of the expression programs are anticorrelated. This, by default, is set on 0. This check is useful when using a small number of modules, since otherwise it might not be easy to generate enough variability to differentiate the branches in the dimensionality reduction step:

```
above_cutoff = (np.max(rel_means[branch]) > rel_exp_cutoff)
parallels = sut.find_parallel(tree, programs, branch)
diverges = sut.diverging_parallel(parallels, rel_means, tree.G, tol=inter_branch_tol)
```

This process results in B matrices with the size $G \times T_b$, where G is the number of expression programs and T_b the length of branch B in pseudotime units. Each row contains the time evolution (T_b steps) of the relative expression of gene g for the current branch.

3.2.3 Getting average gene expression along tree

Translating the relative average expression to absolute expression is straightforward; the matrices of relative expression just need to be scaled by the base expression of each gene (the "100%" of expression they would have when unperturbed by the differentiation procedure that is being simulated):

```
import sim_utils as sut
gene_scale = sut.simulate_base_gene_exp(tree, programs)
Ms = {}
for branch in tree.branches:
    Ms[branch] = np.exp(programs[branch]) * gene_scale
tree.add_genes(Ms)
```

After that, the simulation of the lineage tree is ready and it is time to sample cells from it.

3.3 Sample cells from tree

PROSSTT offers a number of ways to sample cells from the lineage tree:

- sample all points on lineage tree
- sample a pseudotime series
- sample density
- sample particular points of the lineage tree

3.3.1 Sample whole tree

In this mode, all possible pairs of pseudotime and branch t, b on the tree are sampled a number of times. This mode creates simulations that are easier for trajectory inference algorithms to reconstruct, since all points of the lineage tree are available, i.e. there are no gaps that the algorithm must fill on its own.

To sample every point on the tree 3 times:

```
from prosstt import simulation as sim
sim.sample_whole_tree(tree, 3)
```

In the same vein, PROSSTT includes a function to speed up the simulation process; it will generate expression programs, average gene expression and sample the whole tree in one step using default parameters. This makes it easy for users to get a first taste of PROSSTT without having to decide on parameter values or familiarizing themselves with the software first (see the minimal example in the PROSSTT github repo).

3.3.2 Sample pseudotime series

In this mode, PROSSTT simulates a time series experiment given the lineage tree. In a time series experiment a cell population is sampled in different time points along its developmental trajectory. Owing to the asynchrony of cellular differentiation, this means that in every sample (at every time point) there is a mixture of cells from different developmental stages.

In PROSSTT, progress through differentiation is measured by pseudotime. Therefore, the mixture of cells from different developmental stages corresponds to a mixture of cells with different pseudotime. Given a set of sample points, PROSSTT samples from a Gaussian distribution around them with a given standard deviation. For each sampled pseudotime, it picks one of the possible branches randomly.

An example and the corresponding plots are available at the github repository.

3.3.3 Sample density

An additional way to sample cells is to impose a certain density on the lineage tree. Consider transdifferentiations like the one described by Treutlein *et al.* in Nature. Most cells will follow one path, but some could progress into a different cell fate than the intended. The difference in efficiency or in abundance of each branch can be represented by a different density.

Differences in differentiation speed can also be represented by different densities. Imagine a differentiation where cells progress quickly to the cell fate decision point and then linger there, after which they progress quickly to their respective cell fates. For PROSSTT, this would mean having a higher density around the branch point and a lower one towards the end points of the lineage tree, as demonstrated in another example notebook.

3.3.4 Combinations and manual sampling

Density sampling and pseudotime series can be combined; after determining the density of each branch, users can then sample from the tree in a pseudotime series experiment (for an example, see the corresponding jupyter notebook)

Finally, if users are not covered by the available sampling options and want to create something more elaborate, there is always the option to sample pairs of time points and branch assignments from the lineage tree and let PROSSTT simulate the cells for this input:

```
from prosstt import tree
from prosstt import simulation as sim

newick_string = "(A:50,B:50)C:50;"
G = 500
lineage = tree.Tree.from_newick(newick_string, genes=G)
sample_times = np.array([0, 25, 49, 51, 149, 149])
sample_branches = np.array(['C', 'C', 'C', 'B', 'A', 'A'])
sim._sample_data_at_times(lineage, sample_times, branches=sample_branches)
```

3.4 Simulate UMI counts

PROSSTT simulates UMI counts using a negative binomial distribution where the variance of the expression of each gene g depends on its average expression μ_g:

$\sigma_g^2 = \alpha_g \mu^2 + \beta_g \mu_g$

This relationship is preserved through pseudotime; as average expression changes with time, so does the variance, always obeying the same relationship.

$\sigma_g^2(t) = \alpha_g \mu^2(t) + \beta_g \mu_g(t)$

A negative binomial is the distribution of the number of successes in a sequence of i.i.d. Bernoulli trials before a specified number of failures occurs. The negative binomial can be parametrized by its mean and variance or by a pair $p \in (0, 1), r > 0$, where p is the success probability in each Bernoulli trial and r the number of failures. While the negative binomial is originally a discrete probability distribution, it can easily be extended into a continuous one, preserving most of its attributes.

Here we use the implementation of the negative binomial distribution by the *scipy* package, after translating the mean and variance of each distribution to the p, r equivalents.

The gene-specific parameters α_g, β_g are sampled from ranges found in real data. Users can set α_g to 0 and β_g to 1 to have genes with Poisson distributions, or only set α_g to 0 to have genes with scaled Poissonian noise.

CHAPTER

FOUR

NOTES FOR DEVELOPERS

4.1 Building the documentation

Building the documentation requires following additional python packages:

- sphinx
- sphinx_rtd_theme
- sphinx-argparse
- sphinxcontrib-bibtex

After installing the dependencies you can create the html documentation using:

```
cd doc
make html
```

You can find the generated documentation in `build/html`.

4. Noise modelling of scRNA-seq count data

4.1. Introduction

Biochemical reactions are intrinsically stochastic [36, 133], and this causes variation in the observed concentrations of mRNA and protein in cells. In single-cell transcriptomics, this variability is compounded by the capture inefficiencies of experimental protocols. The observed variability is usually referred to as "noise", and separated between biological (the stochasticity of transcription) and technical (caused by capture inefficiencies, batch effects).

The quantification of cell-to-cell variability lies at the heart of most, if not all, analysis methods for single-cell transcriptomics, ranging from clustering, to dimensionality reduction, to pseudotime ordering, to differential gene expression. If noise is not taken into account when calculating pairwise similarity between cells then it can obscure the true relationships between cells.

Analysis of the underlying kinetics of (steady-state) transcription implies that mRNA counts should be Poisson-distributed [7]. It was soon observed that data show a higher dispersion than a Poisson process would support [17, 51, 72]. In response to this Kim and Marioni [88] proposed a beta Poisson model that allowed for on-off transcription. These strategies were complemented by the Bayesian approach of Vallejos *et al.* [177] which allowed propagation of uncertainty from parameter estimates to downstream analysis.

While a Poisson process fundamentally describes the generative process of transcription, the stochasticity of transcriptional bursting introduces additional variability in mRNA levels. This overdispersion can be appropriately modeled by the negative binomial distribution [17, 51, 72], which is now widely accepted. The negative binomial is the simplest appropriate model for discrete overdispersed data, and recently a mechanistic model compatible with the negative binomial distribution was proposed for gene expression. We formed the hypothesis that the variance of the read counts x_g of a gene g depends quadratically on its expectation value μ_g, $\sigma_g^2 = a\mu_g^2 + b\mu_g$. This model corresponds to a Poisson distribution for $a = 0, b = 1$, it describes the so-called "scaled Poissons" for $a = 0, b > 1$ and finally can describe more complex relations with $a > 0, b > 1$.

Many of the tools used for data analysis and visualization still have not made the transition to using negative binomial distributions. Of the dimensionality reduction techniques mentioned in the Introduction, only Monocle and ZINB-WaVE explicitly use negative binomials to quantify pairwise distances. Many tools instead normalize for library size, either by subsampling or by normalizing to the mean/median count number and then take the logarithm of the data plus a pseudocount to avoid zeros. The log-transformed data is then considered approximately Gaussian-distributed by methods such as UMAP, LLE, tSNE, diffusion maps and methods that build on them.

The Gaussian kernel is a good measure of cell-to-cell similarity only if the statistical error due to biological variation and technical sources is approximately constant for all genes, cells, and all

expression strengths. This is fulfilled for strongly expressed genes $\mu_g \gg b/a$, where the quadratic term in the variance $a\mu_g^2$ dominates over the linear term $b\mu_g$ and over the pseudocounts, because we then have $\mathrm{var}(\log(x_g+1)) \approx \mathrm{var}(x_g)/(\mu_g+1)^2 \approx a\mu_g^2/(\mu_g+1)^2 \approx \mathrm{const}$.

Based on published data [51, 72] we expected that typically $b \approx 1 \ldots 2$ and $a \approx 0.1 \ldots 0.2$, and therefore the assumption that the log-transformed data is approximately Gaussian only holds for expected counts much above 10. The variance for genes with lower expression is severely underestimated with this kernel, which means that the read counts of lowly expressed genes are given too much weight in the computation of pairwise distances. That could explain why omitting genes with low expression values from the analysis has often been found advantageous (for example [174, 175]; also see Discussion in [12]). But because only a small fraction of genes have high average expression, omitting the lowly expressed genes discards massive amounts of information, such as the presence or absence of transcription factors.

One of the early goals of my PhD was to investigate the distribution of count data in scRNA-seq experiments and develop a probabilistic noise model that could be used to better describe the data. This in turn would improve downstream analysis by improving dimensionality reduction, clustering, and pseudotime ordering, all indispensable steps of single-cell data analysis.

We planned to determine the posterior distribution of parameters α_g, β_g and c_n for all genes g and cells n by sampling from the joint posterior distribution of all parameters (α_g, β_g, c_n, μ_{kg}, the mixture weights ρ_k and various hyperparameters) using the Hybrid Monte Carlo sampling algorithm [34].

Having obtained the model parameters we could compute the probability $p(\text{same distribution}|\ \mathbf{x}_n, \mathbf{x}_m)$ that the gene count vectors of a pair of cells were generated from the same underlying distribution. This probabilistic measure of pairwise cell similarity could be the input to tSNE, UMAP, diffusion maps, or any comparable method, and lead to any downstream analysis that flows from dimensionality reduction.

Additionally, we could calculate a probabilistic distance measure and use it to perform weighted averaging around each cell n and reduce noise. The weights are simply the probabilities $p(\text{same distribution}|\mathbf{x}_n, \mathbf{x}_m)$. This denoised matrix can be the input to downstream analysis or it can be analyzed again by the same pipeline to produce a denoised probabilistic distance matrix.

4.2. Validating the negative binomial assumption

Inspired by Grün *et al.* [51] we analyzed data from recent publications [51, 72, 202, 65] to see if negative binomials explained variability equally well or better than Poisson or log-normal distributions. For each dataset, after performing cell and gene quality control according to the respective publications, we grouped cells based on either experimental design or the annotation given by the authors, creating clusters of cells that could be considered of the same (sub-)type. This meant that the variability we observed in each cell cluster would be mostly due to noise and not because of biological diversity. This resulted in 52 cell clusters with 10-450 cells each and one cluster with 5 cells. We compared the goodness of fit between the negative binomial, Poisson, and log-normal fits using the Bayes information criterion [147]. The results are visualized in Fig. 4.1 and reflect the results presented by Grün *et al.* [51], who used the χ^2 test. For all cases except the CelSeq2 data, negative binomials describe the underlying distributions better. In all cases a very sizable portion of all genes (at least one third) have

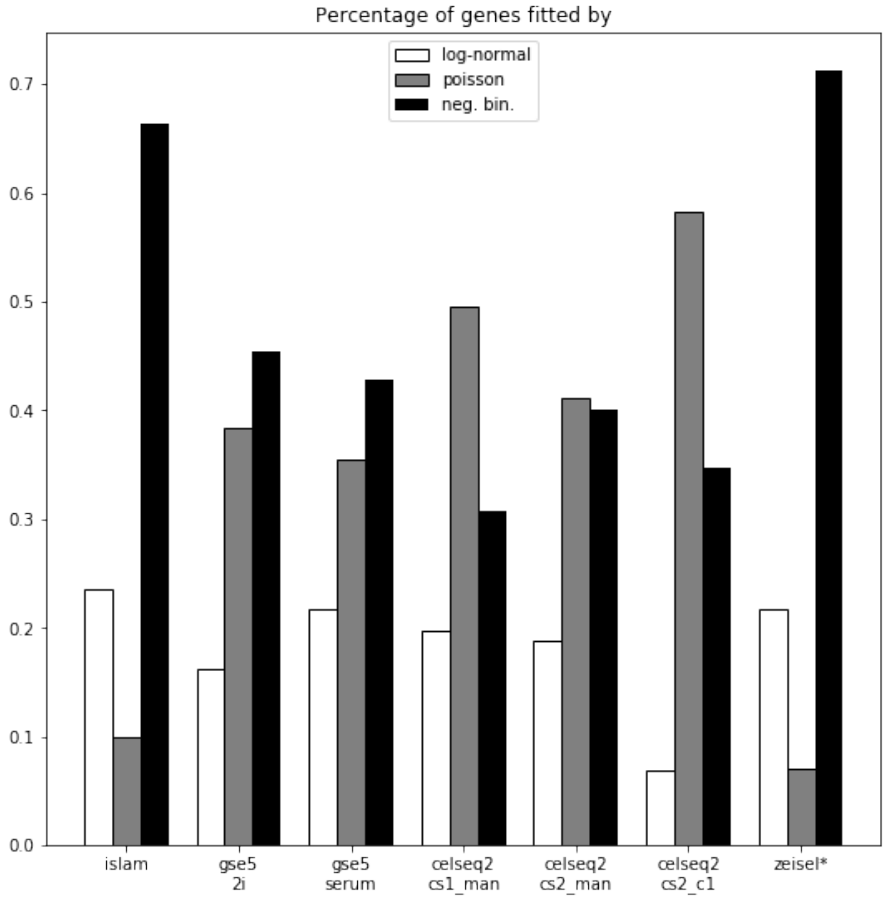

Figure 4.1.: Percentage of genes in each dataset whose distribution was best explained by a log-normal, Poisson or negative binomial, respectively, according to the Bayes information criterion.

count variance that is too high to be described by Poisson distributions.

The relevant analysis can be found on github (`https://github.com/soedinglab/prosstt-scripts`) as well as the supplemental material of PROSSTT, in chapter 3.3, section 3.

4.3. A negative binomial model for scRNA-seq with UMIs

We propose a negative binomial model. For the purpose of estimating the noise, we assume that each of the N cells has counts which are distributed according to a K-component mixture of negative binomial distributions (with $K \approx \sqrt{N}$) and that each component k has a variance that depends on the expectation value μ_{kg} as $\sigma^2_{kg} = c_n(\alpha_g \mu^2_{kg} + b_g \mu_{kg})$, scaled by a cell-specific variance parameter c_n. We propose to determine the posterior distribution of parameters α_g, β_g and c_n for all genes g and cells n by sampling from the joint posterior distribution of all parameters (α_g, β_g, c_n, μ_{kg}, the normally distributed mixture weights ρ_k and various hyperparameters) using the Hybrid Monte Carlo (HMC) sampling algorithm [34].

4.3.1. Notation

Observed:

x_{ng}	number of UMIs or reads for gene $g \in \{1,\dots,G\}$ and cell $n \in \{1,\dots,N\}$

Model parameters:

μ_{kg}	rate parameter of negative binomial for mixture component $k \in \{1,\dots,K\}$ and gene g
s_n	factor describing cell-specific variation in mRNA total content, capture efficiency etc.
α_g, β_g, c_n	gene- and cell-specific dispersion parameters: $\mathrm{var}(x_g) = c_n(\alpha_g \mathrm{E}[x_g]^2 + \beta_g \mathrm{E}[x_g])$
ρ_k	parameters for mixture weight of component k: $p(k) = e^{\rho_k} / \sum_{k'} e^{\rho_{k'}}$
α, β	$p(\mu_{kg}) = \mathrm{Gamma}(\mu_{kg}\vert\alpha,\beta)$
a_0, b_0, Λ	$p(\alpha_g,\beta_g) = \mathcal{N}((\alpha_g,\beta_g)\vert(a_0,b_0),\Lambda)$
σ_s	$p(s_n) = N(\log s_n \vert \log(R_n/\bar{R}), \sigma_s^2)$, with $R_n = \sum_{g=1}^G x_{ng}$ and $\bar{R} = \sum_{n=1}^N R_n/N$
σ_c	$p(c_n) = \mathcal{N}(\log c_n \vert 0, \sigma_c^2)$

4.3.2. Likelihood and prior for all model parameters

We will sample from the posterior distribution over all parameters $\boldsymbol{\theta} = (\boldsymbol{\mu}, \boldsymbol{\rho}, \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \alpha, \beta, a_0, b_0, \Lambda)$,

$$p(\boldsymbol{\theta}|\mathbf{X}) \quad \propto \quad p(\mathbf{X}|\boldsymbol{\theta})\, p(\boldsymbol{\theta}) \tag{4.1}$$

with a likelihood

$$\begin{aligned} p(\mathbf{X}|\boldsymbol{\mu},\boldsymbol{\rho},\boldsymbol{a},\boldsymbol{b},\boldsymbol{c}) &= \prod_{n=1}^N \sum_{k=1}^K p(k)\, p(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{s}, \boldsymbol{a}, \boldsymbol{b}) \qquad (4.2)\\ p(\mathbf{X}|\boldsymbol{\mu},\boldsymbol{\rho},\boldsymbol{a},\boldsymbol{b},\boldsymbol{c}) &= \prod_{n=1}^N \sum_{k=1}^K \frac{e^{\rho_k}}{\sum_{k'} e^{\rho_{k'}}} \prod_{g=1}^G \mathrm{NegBin}(x_{ng}|\mu = s_n\mu_{kg}, \sigma^2 = c_n(\alpha_g\mu^2 + \beta_g\mu)). \end{aligned}$$

The prior distribution is

$$p(\boldsymbol{\theta}) = p(\boldsymbol{s}|\sigma_s)\, p(\boldsymbol{\mu}|\alpha,\beta)\, p(\boldsymbol{a},\boldsymbol{b}|a_0,b_0,\Lambda)\, p(\boldsymbol{c}|\sigma_c^2)\, p(\boldsymbol{\rho})\, p(a_0,b_0)\, p(\sigma_c)\, p(\sigma_s)\, p(\Lambda)\, p(\alpha,\beta) \tag{4.3}$$

$$p(\boldsymbol{s}|\sigma_s) = \prod_{n=1}^N \mathcal{N}(\log s_n|\log(R_n/\bar{R}), \sigma_s), \tag{4.4}$$

$$p(\boldsymbol{\mu}|\alpha,\beta) = \prod_{g=1}^G \prod_{k=1}^K \mathrm{Gamma}(\mu_{kg}|\alpha,\beta), \tag{4.5}$$

$$p(\boldsymbol{a},\boldsymbol{b}|a_0,b_0,\Lambda^{-1}) = \prod_{g=1}^G \mathcal{N}\left(\begin{pmatrix}\log\alpha_g\\ \log\beta_g\end{pmatrix}\middle|\begin{pmatrix}\log a_0\\ \log b_0\end{pmatrix}, \lambda_{aa}\begin{pmatrix}\lambda_{ab}\\ \lambda_{ab}\end{pmatrix}\lambda_{bb}^{-1}\right), \tag{4.6}$$

$$p(\boldsymbol{c}|\sigma_c^2) = \prod_{n=1}^N \mathcal{N}(\log c_n|0,\sigma_c^2), \tag{4.7}$$

$$p(\rho_k) = \mathrm{const.} \tag{4.8}$$

To ensure that the parameters $\mu_{kg}, s_n, a_0, b_0, \lambda_{aa}, \lambda_{bb}, \sigma_s, \sigma_c, \alpha, \beta > 0$ will stay positive during the

sampling, we will parameterize them as $\theta_m = \exp(\xi_m)$, and we actually sample $\xi_m = \log\theta_m$. For notational simplicity in the following we will derive the partial derivatives with respect to the positive parameters θ_m, but the partial derivatives with respect to the ξ_m will be trivially obtained from them by the chain rule,

$$\frac{\partial f(\theta_m)}{\partial \xi_m} = \frac{\partial f(\theta_m)}{\partial \theta_m}\frac{\partial \theta_m}{\partial \xi_m}\exp(\xi_m). \tag{4.9}$$

We chose constant, uninformative priors for the logarithms of these positive parameters,

$$\begin{aligned} p(\log(a_0)) = p(\log(b_0)) \ &= p(\log(\sigma_c)) = p(\log(\sigma_s)) \\ p(\log(\alpha)) = p(\log(\beta)) &= p(\log(\lambda_{aa})) = p(\log(\lambda_{bb}))= \text{const.} \end{aligned} \tag{4.10}$$

In order to ensure that the precision matrix is positive definite, i.e., $|\mathbf{\Lambda}| = \lambda_{aa}\lambda_{bb} - \lambda_{ab}^2 > 0$, its off-diagonal element is parametrized with $\kappa > 0$ as

$$\lambda_{ab} = \exp\left[\frac{1}{2}(\log(\lambda_{aa}) + \log(\lambda_{bb}) - \kappa)\right], \tag{4.11}$$

which yields $|\mathbf{\Lambda}| = \lambda_{aa}\lambda_{bb}\,(1 - e^{-\kappa})$. We again use a constant, non-informative prior on $\log\kappa$.

4.3.3. Hybrid Monte Carlo sampling of parameters

The sampling of the posterior distribution of all parameters can be done quite efficiently using the Hybrid Monte Carlo (HMC) algorithm because we can compute the gradient of the likelihood and hence of the posterior distribution of $\boldsymbol{\theta}$. We do not expect the posterior distribution $p(\boldsymbol{\theta}|\mathbf{X})$ to show high correlations between any pair of variables. Therefore, the parameter space contributing most to the posterior probability will be easily explored with a limited number of samples. We first find the mode $\boldsymbol{\theta}^*$ of the posterior $p(\boldsymbol{\theta}|\mathbf{X}, \boldsymbol{\phi})$ by roughly optimising $mLL(\boldsymbol{\theta}) + \log p(\boldsymbol{\theta})$ and then draw $K_{\text{HMC}} \approx 50$ samples of $\boldsymbol{\theta}$. For each sample we proceed as follows:

(1) Draw a new velocity vector $\boldsymbol{v}_\tau$ of the parameter vector $\boldsymbol{\theta}$ from $\mathcal{N}(\mathbf{0}, \mathbf{I})$ for step τ. The variance of 1 corresponds to an expected kinetic energy of $(D/2)k_BT$ when the mass of the virtual particle is set to 1 and D is the dimensionality of $\boldsymbol{\theta}$. The energy is measured in units of k_BT (hence $k_BT = 1$).

(2) Perform around ten leapfrog steps to integrate the quasi-Newtonian equations of motion in the potential $U(\boldsymbol{\theta}) = -\log p(\mathbf{X}|\boldsymbol{\theta}) - \log p(\boldsymbol{\theta})$ which exerts a force $\boldsymbol{F}(\boldsymbol{\theta}) = -\nabla U(\boldsymbol{\theta})$. In the next section we will show how to efficiently compute $\nabla \log p(\mathbf{X}|\boldsymbol{\theta})$.

(3) Compute the Metropolis-Hastings criterion and add the new parameter vector $\boldsymbol{\theta}_\tau$ if the criterion is fulfilled. Otherwise add the old parameter vector $\boldsymbol{\theta}_{\tau-1}$ another time to the list of samples.

(4) Adapt the time step Δt used in the leapfrog integration.

We iterate these four steps S_{MCMC} times to draw our representative sample $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{S_{\text{HMC}}}$ of parameter vectors from the posterior distribution of $\boldsymbol{\theta}$.

Say we want to achieve a long-term average acceptance rate of α, e.g. 80%. For the adaptation of the time step we have three alternative options. In the first, simpler one we increase the time step $\Delta t \leftarrow \Delta t \times f$ by a factor $f > 1$, e.g. 1.1, if the Metropolis-Hastings criterion was fulfilled, and otherwise decrease it $\Delta t \leftarrow \Delta t / f^4$. In equilibrium, the long-time average of Δt will not change

anymore at a rejection rate of 1 out of 5.

The second option is similar but reduces f over time: $f(\tau) = 1 + \min\{0.1, n^{-0.5}\}$. This ensures that the adaptation will decrease for long integration times and hence that the time reversal will be approximately fulfilled.

The third option aims at a finer regulation of the acceptance probability to a target value of $q_{\text{acc}} \approx 0.8 \ldots 0.9$ by using the posterior probability before and after integration to estimate the size of the error and use it to regulate the adaptation. Let us call r_τ the ratio of the total posterior probability after the τ'th leapfrog integration to the one before integration, hence for $r_\tau \leq 1$ it is the acceptance probability in the Metropolis-Hastings criterion. We keep the time average of r_τ, r_{av}, by updating after each integration step $r_{\text{av}} \leftarrow (r_\tau\, r_{\text{av}}^{T-1})^{1/T}$, with $T \approx 5$. After each integration, we change the time step by a factor $\exp(\delta_{\Delta t})$, $\Delta t \leftarrow \exp(\delta_{\Delta t}) \times \Delta t$, where the factor is chosen as

$$\delta_{\Delta t} = \max\{-\delta_{\text{ceil}}, \min\{+\delta_{\text{ceil}}, 0.2(\log r_{\text{av}} - \log q_{\text{acc}}) + 0.1(\log r_\tau - \log q_{\text{acc}})\}\}. \tag{4.12}$$

By minimizing and maximizing, we limit the factor between $\exp(-\delta_{\text{ceil}})$ and $\exp(\delta_{\text{ceil}})$. As initialization, we choose $\delta_{\text{ceil}} = \log 2/\sqrt{n}$. The two factors 0.2 and 0.1 determine how strongly the time step is corrected due to deviations of $\log r_{\text{av}}$ and r_τ from their target value $\log q_{\text{acc}}$. The two expressions in parentheses implement a proportional and integration (PI) regulator feedback.

4.4. Learning the model parameters

The partial derivatives of the posterior probability were calculated, numerically validated, and implemented (see App. B). However, mixture density models for clustering, such as the Gaussian mixture we use here, usually have many local optima that lie far from the global optimum. To increase the chances of converging to the global optimum we must initialize sufficiently close to it. Additionally, a sanity test is to run the HMC on data generated by the assumed underlying distribution.

A sufficient initialization would be to obtain groupings of very similar cells. If we could make the assumption that cells in each cluster are approximately identical (i.e. have been generated by very similar distributions), then we could use simple heuristics to learn the values of some critical parameters. In the case of differentiations, clustering should be replaced with trajectory inference, grouping together cells at similar developmental stages. We developed MERLoT to achieve the desired level of granularity.

We could not find published tools that could accommodate the underlying distributions of our model, so I developed PROSSTT (see Chapter 3). It is a simulation suite that samples count data from negative binomial distributions with average $s_n \mu_{kg}$ and, setting $\mu = s_n \mu_{kg}$, variance $\sigma^2 = \alpha_g \mu^2 + \beta_g \mu$. The true average expression μ_{kg} changes over pseudotime, simulating differentiation, a change that can also take different directions at cell fate decision points, giving rise to branched trajectories. The only difference to the model described above is that the noise parameter c_n is set to one, an eventual extension for PROSSTT.

4.4.1. Learning the variance

Given an appropriate clustering of the data, there are many options to approximate the average expression μ_{ng} and the variance hyperparameters α_g, β_g, c_n. For simplicity, we often considered cell-specific variance to have negligible effect on the count statistics, setting $c_n = 1$.

Such a clustering can be obtained via spectral clustering, k-means clustering, or any trajectory inference method that produces a tree structure. Indeed, MERLoT in its inception was intended as a method that would produce an initialization for hyperparameter fitting.

Naive polynomial fitting

The simplest approach would be to treat the cells in each group k as identical and estimate the μ_{kg}, σ_{kg} by the empirical average and variance $\hat{\mu}_{kg}, \hat{\sigma}_{kg}$. The cell-wise average μ_{ng} would then be the group average μ_{kg}. For every gene g, a polynomial curve of the form $\sigma^2_{kg} \sim \alpha_g \mu_{kg} + \beta_g \sigma_{kg}$ can be fit over the K different data points, producing robust fits for α_g, β_g.

Simplified negative binomial model

The negative binomial distribution is the discrete distribution of the number of successes in a number of *i.i.d.* Bernoulli trials. The probability mass function is given by

$$\mathrm{NB}(x|r,p) \equiv \Pr(X = x) = \binom{x+r-1}{k} p^x (1-p)^r, \tag{4.13}$$

where $r > 0$ is the number of failures until the experiment is stopped, and $p \in (0,1)$ is the success probability in every trial. It has a mean $\mu = pr/(1-p)$ and variance $\sigma^2 = pr/(1-p)^2$. The Poisson case is obtained for $r \to \infty$, $p \to 0$ and $\mu = pr/(1-p) = \text{const}$.

This definition can be expanded to include continuous count values, e.g. after imputation or normalization, making it useful for scRNA-seq data. It is often named the Polya distribution:

$$\mathrm{NB}(x|r,p) = \frac{\Gamma(r+x)}{x!\Gamma(r)} (1-p)^r p^x \tag{4.14}$$

The relationships of p, r to the mean and variance still hold, so we will use μ and σ as parameters instead:

$$p = 1 - \frac{\mu}{\sigma^2}\,, \qquad\qquad r = \frac{\mu^2}{\sigma^2 - \mu}\,. \tag{4.15}$$

Where we substitute $\sigma^2 = \alpha\mu^2 + b\mu$.

We calculate a simpler version of the model presented in section 4.3, without a Gaussian mixture and priors on the hyperparameters:

$$p(\mathbf{X}|\boldsymbol{\mu}, \boldsymbol{\alpha}, \boldsymbol{\beta}) := \prod_{n=1}^{N} \prod_{g=1}^{G} \mathrm{NB}(x_{ng}|r_{ng}, p_{ng}) \tag{4.16}$$

We compute Euclidean distances between the G-dimensional vectors $\mathbf{x}_n$ after library size normalization and log-transformation. We use the distance matrix to pick the k nearest neighbours of each cell

n and average over them, producing G-dimensional vectors μ_n for each cell n. We can now minimize the negative log-likelihood by taking the partial derivatives with respect to α_g, β_g:

$$\frac{\partial}{\partial \alpha_g} nLL(\alpha_g, \beta_g) := \sum_{n=1}^{N} r_{ng}^2 \psi_0(x_{ng} + r_{ng}) - r_{ng} x_{ng}(1 - p_{ng}) + r_{ng}^2 \log(1 - p_{ng}) - r_{ng}^2 \psi_0(r_{ng}) + \frac{r_{ng}\mu_{ng}^2}{\sigma_{ng}^2} \tag{4.17}$$

$$\frac{\partial}{\partial \beta_g} nLL(\alpha_g, \beta_g) := \sum_{n=1}^{N} \frac{r_{ng}^2 \psi_0(x_{ng} + r_{ng}) - r_{ng} x_{ng}(1 - p_{ng})}{\mu_{ng}} + \frac{r_{ng}^2 \log(1 - p_{ng}) - r_{ng}^2 \psi_0(r_{ng})}{\mu_{ng}} + \frac{r_{ng}\mu_{ng}}{\sigma_{ng}^2} \tag{4.18}$$

We analyzed zebrafish hematopoiesis data by Athanasiadis *et al.* [11], learned $\boldsymbol{\alpha}, \boldsymbol{\beta}$ and calculated distances using a simple Gaussian kernel

$$k(\mathbf{x}_n, \mathbf{x}_m) := \sum_{g=1}^{G} \frac{(x_{ng} - x_{mg})^2}{\sigma_{ng}^2 + \sigma_{mg}^2 + \epsilon}, \tag{4.19}$$

where $\sigma_{ng} = \alpha_g x_{ng}^2 + \beta_g x_{ng}$. We used the resulting pairwise cell-cell distance matrix as input for diffusion maps (Fig. 4.2, top) and compared the embedding to the result obtained when performing typical dimensionality reduction, by calculating the diffusion map of the size-normalized, log-transformed data (Fig. 4.2, bottom).

One downside of this approach is the computational cost; in particular the ψ_0 function is too computationally expensive but needs to be called $\mathcal{O}(n^2)$ times in each evaluation of the derivative function.

Using a Gaussian approximation

Another approach is to relax our modelling assumptions for the first step, and use Gaussian distributions to model noise, assuming all other effects are negligible and the count matrix $\mathbf{X}$ has been normalized to remove library size bias.

$$p(\mathbf{X}|\boldsymbol{\mu}, \boldsymbol{a}, \boldsymbol{b}) := \prod_{n=1}^{N} \prod_{g=1}^{G} \mathcal{N}(\mu_{ng}, \alpha_g \mu_{ng}^2 + \beta_g \mu_{ng}) \tag{4.20}$$

We initialize μ_{ng} with local averages, either obtained via clustering or via the tree. The likelihood can be analytically calculated:

$$nLL(\alpha_g, \beta_g) := \sum_{n=1}^{N} \sum_{g=1}^{G} \ln \sqrt{2\pi(\alpha_g \mu_{ng}^2 + \beta_g \mu_{ng}} + \frac{(x_{ng} - \mu_{ng})^2}{2(\alpha_g \mu_{ng}^2 + \beta_g \mu_{ng})} \tag{4.21}$$

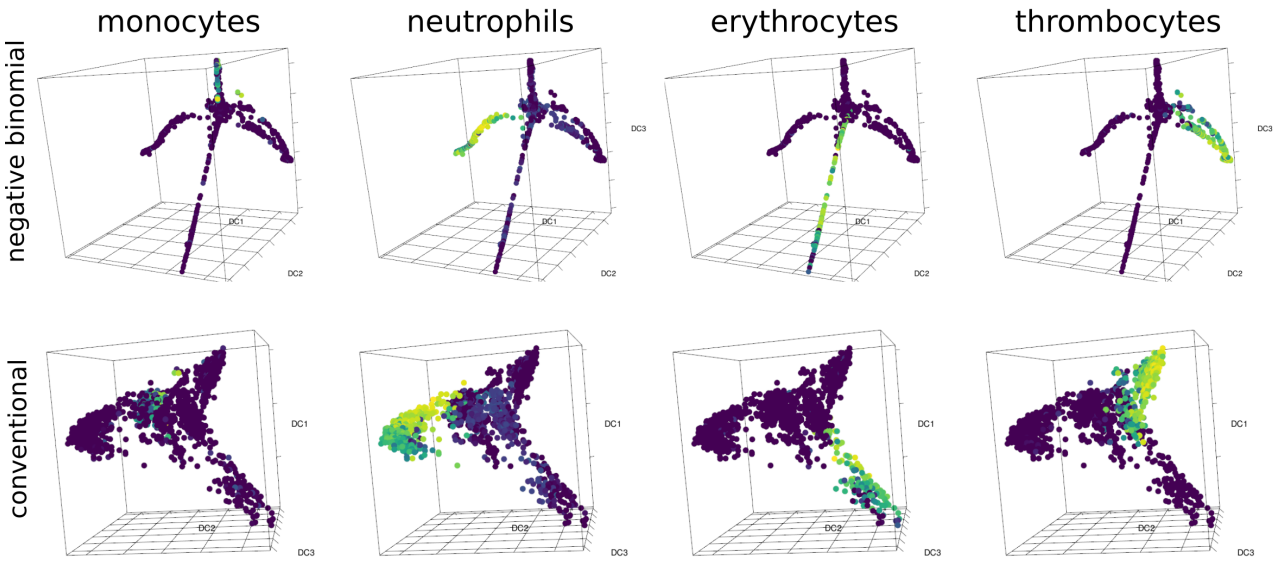

Figure 4.2.: Using variance-weighted distances for diffusion map calculation improves dimensionality reduction for zebrafish hematopoiesis data after learning a simplified negative binomial noise model **Top:** diffusion map of the variance-weighted pairwise cell distance matrix, first three components, colored by respective marker gene expression. **Bottom:** diffusion map of the log-transformed data, first three components, colored by the same marker genes per column. The marker genes characterize different blood cell types: *marco*, for monocytes, *lyz* for neutrophils, *alas2* primarily for erythrocytes, and *itga2b* for thrombocytes. The cell mass that remains unannotated is mostly comprised of hematopoietic stem cell progenitors (also see Fig. 1 in [11]).

The partial derivatives with respect to α_g, β_g are then

$$\frac{\partial}{\partial \alpha_g} LL(\alpha_g, \beta_g) := \sum_{n=1}^{N} \frac{(\alpha_g \mu_{ng}^2 + \beta_g \mu_{ng})\mu_{ng}^2 - (x_{ng} - \mu_{ng})^2 \mu_{ng}^2}{2(\alpha_g \mu_{ng}^2 + \beta_g \mu_{ng})}, \tag{4.22}$$

$$\frac{\partial}{\partial \beta_g} LL(\alpha_g, \beta_g) := \sum_{n=1}^{N} \frac{(\alpha_g \mu_{ng}^2 + \beta_g \mu_{ng})\mu_{ng} - (x_{ng} - \mu_{ng})^2 \mu_{ng}}{2(\alpha_g \mu_{ng}^2 + \beta_g \mu_{ng})} \tag{4.23}$$

This Gaussian approximation was demonstrated to improve diffusion map embeddings of simulated data (lab rotation by Xizhou Zhang, supervised by the author). Such results were not immediately forthcoming in real data. Additionally, this approach is, on the whole, not very efficient, since it fits a probabilistic model only to learn a partial initialization for a bigger optimization problem.

5. Beyond distribution modelling

A lot of effort has been invested in precise modelling of scRNA-seq count data, with the hope of improving cell-cell similarity measures and the analysis pipelines that depend on this quantification. Here we discuss two ideas that make no assumptions about the nature of count data. The first is topology-agnostic averaging; we want to represent each cell as an weighted average of its neighbours by minimizing the variance of the neighbourhood while taking into account the bias of the estimator. Second, we propose a cross validation that allows quantitative assessment of method performance on real datasets, even when no annotation is available.

5.1. Nearest neighbour smoothing with optimal bias-variance trade-off

Let $\mathbf{X} \in \mathbb{N}^{N \times G}$ be the expression matrix of a single-cell RNA-seq experiment, with N cells and G genes captured. Let furthermore NN_i be the indices of the K nearest neighbours of cell i. For brevity we substitute $\sum_j x_{jg} = \sum_{j \in \mathrm{NN}_i x_{jg}}$ and $\sum_g x_{ng} = \sum_{g=1}^{G} x_{ng}$.

Given cell i, its neighbouring cells $j \in \mathrm{NN}_i$, and their expression profiles $x_{jg} \in X$, we want to find weights $w_{ij} \in [0, 1]$ such that

$$\widetilde{x}_{ig} := \frac{\sum_j w_{ij} x_{jg}}{\sum_j w_{ij}} \tag{5.1}$$

is an "optimal" (smoothed) estimator of x_{ig}. In particular, the weights $w_{ij} \in [0, 1]$ should minimize the sum of the bias and the variance of the estimator, and each cell i is considered its own nearest neighbour. The matrix of all w_{ij} is $W \in \mathbb{R}^{N \times K}$, and the i-th row of that matrix is W_i.

The *bias* of the estimator is

$$\mathrm{bias}(\vec{W}_i) := \frac{1}{2} \sum_g \frac{(x_{ig} - \widetilde{x}_{ig})^2}{\widetilde{\sigma}_{ig}^2} \tag{5.2}$$

where $\widetilde{\sigma}_{ig}^2 := \frac{1}{\sum_j w_{ij}} \sum_j w_{ij} (x_{jg} - \widetilde{x}_{ig})^2$ is the weighted empirical variance of the estimator. Each summand is related to the Gaussian probability that x_{ig} belongs to a distribution $\mathcal{N}(\widetilde{x}_{ig}, \widetilde{\sigma}_{ig}^2)$. The sum over G reflects how well $\widetilde{\mathbf{x}}_i$ predicts $\mathbf{x}_i$ by taking into account the variance of the neighbourhood of $\mathbf{x}_i$. This term quantifies how far the weighted average is from the cell it represents.

The *variance* is

$$\mathrm{var}(\vec{W}_i) := \frac{1}{2} \sum_g \frac{\mathrm{var}(\widetilde{x}_{ig})}{\widetilde{\sigma}_{ig}^2} \tag{5.3}$$

With $\mathrm{var}(\widetilde{x}_{ig}) = \sum_j \frac{w_{ij}^2}{(\sum_j w_{ij})^2} \mathrm{var}(x_{jg}) \approx \widetilde{\sigma}_{ig}^2 \frac{\sum_j w_{ij}^2}{(\sum_j w_{ij})^2}$, this becomes

$$\begin{aligned} \text{var}(\vec{W}_i) &= \frac{1}{2}\sum_g \frac{\sum_j w_{ij}^2}{(\sum_j w_{ij})^2} \\ &= \frac{1}{2} G \frac{\sum_j w_{ij}^2}{(\sum_j w_{ij})^2} \end{aligned} \tag{5.4}$$

In the bias equation we are describing the estimator for $\mathbf{x}_i$ as a Gaussian cloud centered on $\widetilde{\mathbf{x}}_i$ with a shape of $\widetilde{\boldsymbol{\sigma}}_i^2$. In the variance equation we are comparing our uncertainty about the location of the estimator with the size of its Gaussian cloud; we quantify how well we can estimate the weighted average. It becomes clear to see that this estimation will improve with more cells (i.e. higher values of K).

For each cell i we optimize

$$\text{bias}(\vec{W}_i) + \text{var}(\vec{W}_i) \underset{\vec{W}_i}{\longrightarrow} min \tag{5.5}$$

The bias and variance have their individual optima in different directions; whereas in the variance term we profit from having higher values of K, higher values of K also mean that the weighted average will be closer to the global average, and so move further away from the local neighbourhoods that the bias term describes.

The partial derivatives can be calculated analytically and the fitting can be done with any gradient-based optimizer, such as the L-BFGS-B algorithm. We will be parametrizing $w_{ij} = e^{-\omega_{ij}}$ to ensure the weights are all positive.

The partial derivatives were calculated (see Appendix C), numerically validated, and implemented efficiently in C. Preliminary results have been encouraging, and we plan to continue work on this chapter after thesis submission. One possible modification would be to simplify the definition of $\widetilde{\sigma}_{ig}^2$ to $\frac{1}{K}\sum_j (x_{jg} - \widetilde{x}_{ig})^2$.

5.2. Cross validation for gene expression imputation

Next to the methods for trajectory inference, a multitude of methods have been designed for the imputation of scRNA-seq count data, grouping similar cells together and transferring information between them [6, 38, 69, 97, 99, 100, 179, 186]. To assess performance and facilitate comparison between methods, Li and Li [97] propose three tests: imputation of ERCC spike-in concentrations [77], imputation of cell-cycle gene levels on staged cells [20], and imputation of simulated data. Other options, as mentioned by Eraslan *et al.* [38], are to compare between bulk and scRNA-seq data of the same subpopulations or compare between scRNA-seq and proteomics data.

At the same time, trajectory inference methods, by ordering cells in pseudotime and arranging them according to cell fate decisions also produce imputation predictions. Some do so explicitly, such as MERLoT [123], which uses the high-dimensional tree fitting to produce denoised expression profiles. However, all trajectory inference methods allow visualization of gene expression change over time, usually representing individual cells as points and plotting a smooth curve over them that represents the "true" or "average" expression.

Here we propose a complementary approach for imputation assessment intended especially for

scRNA-seq data from differentiation experiments. The input is a matrix $\mathbf{X} \in \mathbb{R}^{N \times G}$. This can be either with or without imputation via any method (see above paragraph or previous section). We reconstruct the trajectory, e.g. with MERLoT. We partition the set of genes $\mathcal{G}$ to two subsets $\mathcal{A}_1, \mathcal{A}_2$ of equal size. We assign each cell n to the reconstructed trajectory using either $\mathcal{A}_1$ or $\mathcal{A}_2$. This creates two projections with imputed expression values, $\widetilde{x}_n^{(1)}$ and $\widetilde{x}_n^{(2)}$.

Having obtained the projections for each cell we can calculate the robustness of the assignments:

$$\widetilde{\sigma}_g^2 := \frac{1}{2N} \sum_{n=1}^{N} (\widetilde{x}_{ng}^{(1)} - \widetilde{x}_{ng}^{(2)})^2. \tag{5.6}$$

Ideally the two projections of n are close on the tree, and hence will have very similar imputed expression profiles. We compare this imputation variance with the global variance of gene g:

$$\sigma_g^2 := \frac{1}{N} \sum_{n=1}^{N} (x_{ng} - \bar{x}_g)^2, \tag{5.7}$$

where $\bar{x}_g$ is the global mean of gene g. We can now compare the imputation variance to the global variance, for a small set (10-100) of the most tree-informative genes $\mathcal{T}$:

$$\frac{1}{|\mathcal{T}|} \sum_{g \in \mathcal{T}} \frac{\widetilde{\sigma}_g^2}{\sigma_g^2} \tag{5.8}$$

This score can be repeated for many different partitions of $\mathcal{G}$ to create a bootstrapped estimation of performance with confidence intervals.

6. Generation of a mouse ovary cell atlas with single-cell sequencing

6.1. Introduction

It is widely held that in mammals the oocytes pool is already established during fetal development and no more eggs are produced postnatally [47]. This belief was corroborated by lack of *de novo* mutations in the maternal germline but challenged by the apparent renewal of ovarian follicles [78, 79] (the structures that form around developing oocytes) in postnatal mouse ovaries. Furthermore, it is known that the number and quality of eggs decrease with maternal age [8], and that in pathological cases there can be a loss of ovary function before age 40, a condition named primary ovarian insufficiency [117]. A comprehensive overview of the cellular changes during ovary development and maternal aging could provide new insights.

With this aim, the Meiosis lab, led by Melina Schuh (Max Planck Institute for Biophysical Chemistry, Göttingen, Germany) and considered a world expert in mammalian oocytes, planned the generation of a mouse ovary cell atlas with scRNA-seq, going from fetal gonads to aged mice ovaries. In order to study ovary development they want to sample perinatal time points, starting at embryonic day 12 ("E12" sample), where sex can be morphologically distinguished, and going to second postnatal day ("P2"), possibly finding new cell type subpopulations such as the "ovarian stem cells", and identifying novel genes with a function in oocyte development. In order to study the effects of maternal aging on ovaries and find novel aging markers they want to compare the newborn mice to young (10 weeks old, "10w"), mid-aged (35 weeks old, "35w") and aged ($\geq$ 58 weeks old) females. Additionally, the aged females will be either never mated (aged females, "af") or always mated (aged breeding females, "abf"), where ovary depletion might create a link to primary ovarian insufficiency. The lab can use locally established methods for whole mount microscopy and immunohistochemistry to validate interesting results.

I was consulted in the planning stage, and contributed with data analysis and technical troubleshooting in both the pilot stage and the first round of data generation. In the pilot stage of the experiment, E13 and 10w were sequenced. The E13 timepoint is particularly interesting, since it is the onset of meiosis. Unfortunately, due to problems in cell dissociation, the 10w data were of very low quality. The first round of data generation contained timepoints E12, E15, E18 (immediately prenatal), P2, a repetition of 10w, abf and af.

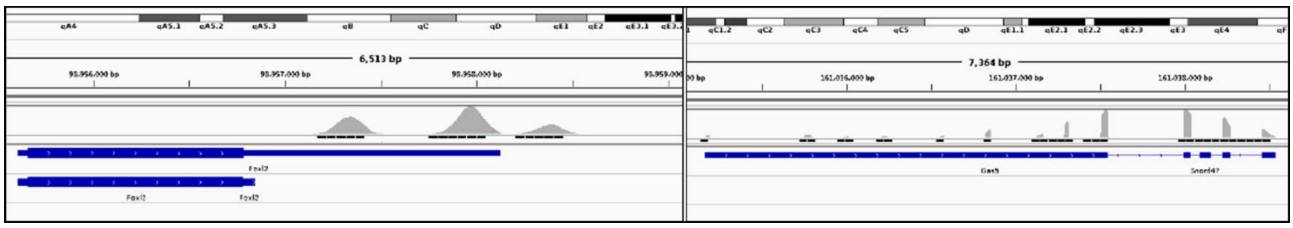

Figure 6.1.: Screenshot from the genome browser that showcases annotation problems using two established marker genes, *Foxl2* (left) and *Gas5* (right). The checkered bar in the top and the numbers below show location in the genome. The gray accumulations are the positions that reads have mapped onto. The dark blue lines on the bottom are genome annotation; the upper one is RefSeq [119] annotation and the lower one is the mouse annotation file from 10x Genomics. *Foxl2* has a long 3' UTR (thinner line in RefSeq annotation), that is absent in the 10x annotation. Since all reads fall into the 3' UTR they will be ignored. *Gas5* is annotated as a "processed pseudogene" by RefSeq, an annotation that is completely excluded in 10x. The mapped reads create the impression of an exonic region that goes into the next annotated region, small nucleolar RNA *Snord47*.

6.2. Pilot stage: E13, 10w

Our analysis was confined to the E13 sample since the 10w sample was of low quality. Alignment to the mouse genome (Ensembl, GRCm38.93), filtering, barcode counting and UMI counting was performed using CellRanger, the tool provided by 10x Genomics, the company that developed the experimental protocol. We aggregated the data from the biological replicates using CellRanger and Seurat [22], and looked for the expression of marker genes.

Analysis by marker gene levels showed that the E13 data contained two large clusters of somatic cells, blood cells, and a large cluster of pluripotent germ cells. Further analysis suggested that known subgroups of somatic cells were present as expected. We encountered a lot of problems with the annotation; many genes appear with different names in the literature and in the genome annotation. Additionally, some genes the literature suggested as markers could not be found in the data even under alternative names.

6.3. First round of data gathering: E12, E15, E18, P2, 35w

In the E15 data we noticed for the first time pronounced differences between the two biological replicates. I performed batch correction on the E15 data using different approaches (mutual nearest neighbours [60], scater [111], Seurat) and proposed a pipeline based on Seurat (version 2). After batch correction the data overlapped quite well, as expected from biological replicates. This demonstrated that the CellRanger software was inadequate for more than rudimentary analysis, and shouldn't be relied on. Instead, I developed a pre-processing, aggregation, and analysis (e.g. differential gene expression detection) pipeline based on Seurat.

At this stage, we also revisited the missing gene problem, and realized that some of the marker genes that we looked for (*Foxl2*, a marker for granulosa cell precursors, and *Gas5*, an aging marker) were actually present in the data (i.e. there were reads that mapped to the correct genomic region), but were "lost" due to annotation errors. In particular, the 3'-UTR of *Foxl2* was unannotated, and the *Gas5* region was marked as a "processed pseudogene", hence the reads were discarded. The only way to overcome this would be to curate the annotation gtf file (Gene set file, containing annotations for

coding and non-coding regions) by manually adding the problematic regions. We verified our approach by adding the correct entry for *Gas5* to the annotation and then performing alignment, filtering and counting again.

6.4. Outlook

The next part of data gathering was planned for the summer months of 2019, overlapping with thesis writing and my thesis defense. However, once all data is available, it will be possible to integrate the different time points and study perinatal ovary development in single cell resolution.

7. Conclusion and outlook

The explosive growth in scRNA-seq data has made it possible to study highly complex processes, such as organogenesis or whole organism development [24, 42, 18] on the single-cell level. These advances have caused a similar explosion in demand for computational analysis options [142, 201]. The novel algorithms and ideas proposed in this work contribute towards the improvement of trajectory inference, method performance assessment, and modelling of count data.

MERLoT is a novel trajectory inference method that can reconstruct complex lineage trees by smoothing a robust initial tree and embedding into gene expression space. The embedding step is what sets MERLoT apart from most other methods (for example TSCAN [76] and Slingshot [161], which connect cell clusters and refine this rough tree), translating the lineage tree to gene expression space, facilitating downstream analysis, and improving the interpolated expression profiles of the lineage tree nodes. As the transcriptomics field continues advancing, bigger and more complex datasets are generated. MERLoT has already demonstrated, on real [45] and simulated [123] data, that it is well-equipped to deal with these challenges.

The last years have also produced significant advances in the field of visualization and dimensionality reduction. General purpose methods like PHATE [115] and UMAP [112] have been adapted while methods like URD [41] and PAGA [193] have been developed specifically for single cell transcriptomics visualization. While more conceptual representations like PAGA are useful, they might not be very well-suited to represent transitions, such as expression gradients within a cell type, in the same way that methods that visualize on the cell level can. MERLoT is not tied to a specific dimensionality reduction method, and can be applied on top of any coordinates, but it also offers conceptual visualizations for trees in higher dimensions.

Most dimensionality reduction algorithms and visualization techniques translate transcriptome similarity to distance on a reduced manifold. The quality of the mapping is directly related to how well these similarities are quantified. The usual choice, Euclidean distances on count or transformed data, suffer from the curse of dimensionality. A probabilistic distance kernel that uses an appropriate statistical model to represent the data would overcome this problem. We proposed a Gaussian mixture of negative binomial distributions to describe single cell count data from differentiation experiments. Using this model to define a distance kernel is straightforward, but learning the model hyperparameters is not (also see Risso *et al.* [136], who train a similar model). While negative binomial distributions have long been considered an appropriate model for count data [51, 72], they have not received as much attention in the computational side. The recent success of methods that use negative binomial distributions [130, 136, 38] and the proposal of a mechanistic transcription model [7] signal that maybe this trend is at an end, and soon we might see negative binomial kernels used in diffusion maps, UMAP, or t-SNE.

I am currently working on two approaches that don't make explicit model assumptions. First, we

aim to decrease noise by weighted local averaging. Instead of only imputing missing values [97] or building a model and applying it to each cell [69], we transfer information from nearby cells, a much milder form of imputation. Since we don't correct each gene separately we preserve the non-linear relationships between the genes. This approach is also more robust to the choice of distance measure; if inappropriate cells are suggested as neighbours they will just receive very low contribution weights. Second, we are interested in comparing our imputation procedure and others [38, 186, 69, 97, 179] using an adaptation of cross validation for scRNA-seq. In particular, we want to use the robustness of trajectory inference as a measure of imputation quality. This would be a departure from established practice. To date, imputation quality is usually demonstrated by introducing zeros or noise in an expression matrix and trying to recover the original expression values [38] or comparing dimensionality reduction and gene expression plots before and after imputation.

A major development in the field of lineage reconstruction was the introduction of lineage tracing for developmental processes. Various techniques can be used to tag cells in developing embryos with random CRISPR-Cas9 scars starting at the zygote, so that entire cell populations can later be traced back to their ancestor cell via their scar barcodes [157, 83, 113]. Lineage tracing with CRISPR has not been widely adopted as of yet, but it is easy to imagine how much it could benefit trajectory inference. Achieving regular and predictable CRISPR scarring, e.g. once every cell division, would constitute an internal differentiation timer, providing a natural alternative to pseudotime.

Without such foolproof internal consistency markers for pseudotime or cell identity, the only way to validate method performance quantitatively is to use simulated data. In the early years of single-cell transcriptomics a lot of simulation methods were developed. They aimed to simulate different groups of cells with differential expression between them and account for other factors, such as dropout and batch effect [102, 101, 177]. Splatter [200] was a more sophisticated approach that included previous models, and could also simulate trajectories. However, the trajectory module of Splatter produced artifacts, as we discovered during the benchmarking of MERLoT [123]. PROSSTT addresses the need for simulated data of complex differentiation trajectories. Its flexible variance model covers a variety of different count distributions proposed for scRNA-seq data, and its model of gene expression change over pseudotime creates rich correlations between the different genes, mimicking the relationships between genes in real data. We think PROSSTT fills an important niche, and can be a valuable tool for the development of trajectory inference tools.

The increase in scRNA-seq throughput has brought the once lofty goal of dissecting gene regulatory networks within reach [55] aided considerably by the advances in single cell epigenetics, and methods to infer networks from single-cell data are being developed [114, 21]. Method development would be greatly facilitated by the existence of data with known regulatory relationships, real or simulated, and while it is possible to simulate simpler regulatory networks and the corresponding thermodynamics [142], this approach doesn't scale beyond a small set of predetermined topologies. Integrating gene regulation to PROSSTT could be an interesting expansion.

Single cell transcriptomics has gone from an experimental breakthrough to widely applied standard technique in the span of ten years. This was accompanied by corresponding advances in computational analysis. Yet, despite the progress, there is still room for improvement: challenges like noise modelling, batch effect removal, and trajectory inference, particularly on large datasets, remain, while new ones, like incorporation of spatial information, and integration of data from multiple sources and data of

different types, are emerging. We hope that the tools and ideas described in this thesis will benefit the community and help lead to new and interesting discoveries.

Appendices

A. Representative single-cell sequencing methods in the last ten years

The selected publications overlap with those in [162] up to March 2017. After that, publications were selected from a literature search. These publications are by far not the total of available single-cell studies. It can be claimed without exaggeration that experiments with a few thousands of cells are the current standard in the field right now. Suspiciously round numbers are not exact but correspond to the authors' descriptions of the datasets.

Year	**Month**	**#Cells**	**Citation**
2009	5	1	Tang *et al.* [163]
2010	5	34	Tang *et al.* [164]
2011	5	85	Islam *et al.* [71]
2011	6	24	Tang *et al.* [165]
2012	8	96	Hashimshony *et al.* [64]
2012	8	12	Ramsköld *et al.* [132]
2012	10	6	Brouilette *et al.* [19]
2013	4	77	Sasagawa *et al.* [143]
2013	4	10	Magnúsdóttir *et al.* [105]
2013	6	18	Shalek *et al.* [150]
2013	7	124	Yan *et al.* [198]
2013	11	100	Brennecke *et al.* [17]
2013	11	68	Picelli *et al.* [126]
2013	12	18	Grindberg *et al.* [50]
2013	12	15	Marinov *et al.* [108]
2014	1	100	Wu *et al.* [195]
2014	2	192	Islam *et al.* [72]
2014	2	4,000	Jaitin *et al.* [73]
2014	5	400	Treutlein *et al.* [174]
2014	6	1,800	Shalek *et al.* [151]
2014	6	768	Patel *et al.* [124]
2014	12	415	Kumar *et al.* [92]
2015	2	15,000	Fan *et al.* [39]
2015	4	470	Darmanis *et al.* [29]
2015	4	3,000	Zeisel *et al.* [202]
2015	5	44,808	Macosko *et al.* [104]

2015	5	9,000	Klein *et al.* [89]
2015	5	400	Bose *et al.* [15]
2015	10	869	Kolodziejczyk *et al.* [90]
2015	11	9,000	Paul *et al.* [125]
2016	3	1,682	Habib *et al.* [57]
2016	4	4,600	Tirosh *et al.* [171]
2016	6	5,000	Marques *et al.* [109]
2016	6	3,000	Lake *et al.* [95]
2016	7	1,800	Tasic *et al.* [167]
2016	9	2,200	Yuan and Sims [199]
2016	9	1,422	Joost *et al.* [81]
2016	10	3,000	Vickovic *et al.* [183]
2016	10	6,100	La Manno *et al.* [93]
2016	10	12,200	Muraro *et al.* [116]
2016	12	200,000	Dixit *et al.* [33]
2016	12	50,000	Jaitin *et al.* [74]
2016	12	86,000	Adamson *et al.* [2]
2017	1	250,000	Zheng *et al.* [205]
2017	1	5,905	Datlinger *et al.* [31]
2017	1	1,369	Alles *et al.* [5]
2017	2	4,638	Gierahn *et al.* [46]
2017	2	42,000	Cao *et al.* [24]
2017	2	1,500	Halpern *et al.* [61]
2017	3	1,486	Guillaumet-Adkins *et al.* [54]
2017	3	14,391	Venteicher *et al.* [182]
2017	3	14,000	Chen *et al.* [25]
2017	4	2,400	Villani *et al.* [184]
2017	6	2,167	Li *et al.* [96]
2017	7	2,616	Stoeckius *et al.* [160]
2017	8	29,543	Habib *et al.* [58]
2017	8	20,424	Adam *et al.* [1]
2017	10	3,589	Darmanis *et al.* [30]
2017	10	20,679	Wu *et al.* [197]
2017	10	7,975	Karaiskos *et al.* [86]
2017	11	18,000	Poran *et al.* [129]
2017	11	53,193	Haber *et al.* [56]
2017	12	11,759	Schelker *et al.* [145]
2017	12	18,000	Hu *et al.* [68]
2017	12	14,104	Kang *et al.* [84]
2018	1	5,454	Hochgerner *et al.* [67]
2018	1	10,519	Skelly *et al.* [156]

2018	2	20,387	Stephenson *et al.* [158]
2018	2	400	Stevant *et al.* [159]
2018	2	8,000	Torre *et al.* [172]
2018	2	65,000	Han *et al.* [63]
2018	3	2,300	Zhong *et al.* [206]
2018	3	60,000	Raj *et al.* [131]
2018	3	3,655	Shnayder *et al.* [154]
2018	4	156,049	Rosenberg *et al.* [138]
2018	4	2458	Filbin *et al.* [43]
2018	4	25,000	van der Wijst *et al.* [178]
2018	4	13,000	Pandey *et al.* [120]
2018	4	2,880	Herman *et al.* [66]
2018	4	10,000	Alemany *et al.* [4]
2018	5	11,888	Sebé-Pedrós *et al.* [148]
2018	5	15,539	Rodda *et al.* [137]
2018	5	25,790	Nguyen *et al.* [118]
2018	5	57,979	Park *et al.* [122]
2018	5	6,862	Kim *et al.* [87]
2018	5	21,612	Plass *et al.* [128]
2018	6	90,000	Wagner *et al.* [185]
2018	6	38,731	Farrell *et al.* [42]
2018	6	4,000	Fan *et al.* [40]
2018	6	136,966	Briggs *et al.* [18]
2018	7	3,142	Wang *et al.* [190]
2018	7	10,000	Sasagawa *et al.* [144]
2018	8	34,188	Andor *et al.* [9]
2018	8	509,876	Zeisel *et al.* [203]
2018	9	1,500	Karaayvaz *et al.* [85]
2018	9	35,000	Green *et al.* [49]
2018	12	1,976	Collin *et al.* [28]
2018	12	76,070	Reyfman *et al.* [135]
2019	1	76,000	Hammond *et al.* [62]
2019	1	1,464	Ji *et al.* [75]
2019	1	3,461	Jordão *et al.* [82]
2019	1	36,931	Braga *et al.* [16]
2019	2	1,106	Tiklová *et al.* [170]
2019	2	10,000	Ryu *et al.* [141]
2019	2	315,000	Schiebinger *et al.* [146]
2019	3	38,410	van Galen *et al.* [180]
2019	3	3,152	Francesconi *et al.* [44]
2019	4	1,088	Wen *et al.* [192]
2019	4	2,607	Sharma *et al.* [152]

2019	4	17,374	Tikhonova *et al.* [169]
2019	5	12,000	Shulse *et al.* [155]

B. Partial derivatives of posterior probability for noise model

All necessary partial derivatives are presented here. These are:

- For the prior (see Eq. 4.3):

  since the derivatives of the logs of all probabilities to the right of $p(\boldsymbol{\rho})$ are constant per definition, we only need the following derivatives:

$$\begin{aligned} p(\boldsymbol{s}|\sigma_s) &\rightarrow s_n, \sigma_s \\ p(\boldsymbol{\mu}|\alpha,\beta) &\rightarrow \mu_{k,g}, \alpha, \beta \\ p(\boldsymbol{a},\boldsymbol{b}|a_0,b_0,\Lambda) &\rightarrow a_g, b_g, a_0, b_0, \lambda_{aa}, \lambda_{bb}, \kappa \\ p(\boldsymbol{c}|\sigma_c^2) &\rightarrow c_n, \sigma_c \end{aligned} \tag{B.1}$$

  and thus we calculate:

$$\begin{aligned} \log p(\boldsymbol{s}|\sigma_s) &= \log \prod_{n=1}^{N} \mathcal{N}(\log s_n | \log(R_n/\bar{R}), \sigma_s) \\ &= \sum_{n=1}^{N} \log \mathcal{N}(\log s_n | \log(R_n/\bar{R}), \sigma_s) \\ &= \sum_{n=1}^{N} \log \mathcal{N}(\log s_n | \log(R_n/\bar{R}), \sigma_s) \\ &= \sum_{n=1}^{N} \left(\log \frac{1}{\sqrt{2\pi\sigma_c^2}} + \log exp(-\frac{(\log s_n - \log \frac{R_n}{\bar{R}})^2}{2\sigma_c^2}) \right) \\ &= \sum_{n=1}^{N} \left(-\log(\sqrt{2\pi\sigma_c^2}) - \frac{(\log s_n - \log \frac{R_n}{\bar{R}})^2}{2\sigma_c^2} \right) \\ &= \sum_{n=1}^{N} \left(-\log(\sqrt{2\pi\sigma_c^2}) - \frac{1}{2\sigma_c^2}(\log s_n - \log \frac{R_n}{\bar{R}})^2 \right) \\ &= \sum_{n=1}^{N} \left(-\log\sqrt{2\pi} - \log\sigma_c - \frac{1}{2\sigma_c^2}(\log s_n - \log \frac{R_n}{\bar{R}})^2 \right) \end{aligned} \tag{B.2}$$

$$\boxed{\frac{\partial}{\partial s_n}\log p(\boldsymbol{s}|\sigma_s) = \frac{1}{s_n\sigma_c^2}(\log\frac{R_n}{\bar{R}} - \log s_n)} \tag{B.3}$$

$$\begin{aligned}\frac{\partial}{\partial \sigma_c}\log p(\boldsymbol{s}|\sigma_s) &= \sum_{n=1}^{N}\left(-\frac{1}{\sigma_c} + \frac{1}{\sigma_c^3}(\log\frac{R_n}{\bar{R}} - \log s_n)^2\right) \\ &= -\frac{N}{\sigma_c} + \frac{1}{\sigma_c^3}\sum_{n=1}^{N}(\log\frac{R_n}{\bar{R}} - \log s_n)^2 \end{aligned} \tag{B.4}$$

$$\boxed{\frac{\partial}{\partial \sigma_c}\log p(\boldsymbol{s}|\sigma_s) = -\frac{N}{\sigma_c} + \frac{1}{\sigma_c^3}\sum_{n=1}^{N}(\log\frac{R_n}{\bar{R}} - \log s_n)^2} \tag{B.5}$$

$$\begin{aligned}\log p(\boldsymbol{\mu}|\alpha,\beta) &= \log\prod_{g=1}^{G}\prod_{k=1}^{K}\mathrm{Gamma}(\mu_{kg}|\alpha,\beta) \\ &= \sum_{g=1}^{G}\sum_{k=1}^{K}\log\mathrm{Gamma}(\mu_{kg}|\alpha,\beta) \\ &= \sum_{g=1}^{G}\sum_{k=1}^{K}\log\frac{\beta^\alpha\mu_{kg}^{\alpha-1}e^{-\mu_{kg}\beta}}{\Gamma(\alpha)} \\ &= \sum_{g=1}^{G}\sum_{k=1}^{K}\left((\log\beta^\alpha + \log\mu_{kg}^{\alpha-1} + \log e^{-\mu_{kg}\beta} - \log\Gamma(\alpha))\right) \\ &= \sum_{g=1}^{G}\sum_{k=1}^{K}(\alpha\log\beta + (\alpha-1)\log\mu_{kg} - \mu_{kg}\beta - \log\Gamma(\alpha)) \end{aligned} \tag{B.6}$$

$$\begin{aligned}\frac{\partial}{\partial\alpha}\log p(\boldsymbol{\mu}|\alpha,\beta) &= \sum_{g=1}^{G}\sum_{k=1}^{K}\frac{\partial}{\partial\alpha}(\alpha\log\beta + (\alpha-1)\log\mu_{kg} - \mu_{kg}\beta - \log\Gamma(\alpha)) \\ &= \sum_{g=1}^{G}\sum_{k=1}^{K}(\log\beta + \log\mu_{kg} - \frac{1}{\Gamma(\alpha)}\Gamma(\alpha)\psi^0(\alpha)) \\ &= \sum_{g=1}^{G}\sum_{k=1}^{K}(\log\beta + \log\mu_{kg} - \psi^0(\alpha)) \end{aligned} \tag{B.7}$$

$$\boxed{\frac{\partial}{\partial\alpha}\log p(\boldsymbol{\mu}|\alpha,\beta) = KG(\log\beta - \psi^0(\alpha)) + \sum_{g=1}^{G}\sum_{k=1}^{K}\log\mu_{kg}} \tag{B.8}$$

$$\begin{aligned}\frac{\partial}{\partial\beta}\log p(\boldsymbol{\mu}|\alpha,\beta) &= \sum_{g=1}^{G}\sum_{k=1}^{K}\frac{\partial}{\partial\beta}(\alpha\log\beta+(\alpha-1)\log\mu_{kg}-\mu_{kg}\beta-\log\Gamma(\alpha)) \\ &= \sum_{g=1}^{G}\sum_{k=1}^{K}(\frac{\alpha}{\beta}-\mu_{kg}) \qquad \text{(B.9)}\end{aligned}$$

$$\boxed{\frac{\partial}{\partial\beta}\log p(\boldsymbol{\mu}|\alpha,\beta)=KG\frac{\alpha}{\beta}-\sum_{g=1}^{G}\sum_{k=1}^{K}\mu_{kg}} \qquad \text{(B.10)}$$

$$\begin{aligned}\frac{\partial}{\partial\mu_{kg}}\log p(\boldsymbol{\mu}|\alpha,\beta) &= \frac{\partial}{\partial\mu_{kg}}\sum_{g=1}^{G}\sum_{k=1}^{K}(\alpha\log\beta+(\alpha-1)\log\mu_{kg}-\mu_{kg}\beta-\log\Gamma(\alpha)) \\ &= \frac{\partial}{\partial\mu_{kg}}(\alpha\log\beta+(\alpha-1)\log\mu_{kg}-\mu_{kg}\beta-\log\Gamma(\alpha)) \\ &= \frac{(\alpha-1)}{\mu_{kg}}-\beta \qquad \text{(B.11)}\end{aligned}$$

$$\boxed{\frac{\partial}{\partial\mu_{kg}}\log p(\boldsymbol{\mu}|\alpha,\beta)=\frac{(\alpha-1)}{\mu_{kg}}-\beta} \qquad \text{(B.12)}$$

$$
\begin{aligned}
\log p(\boldsymbol{a},\boldsymbol{b}|a_0,b_0,\Lambda^{-1}) &= \log\prod_{g=1}^{G}\mathcal{N}((\log a_g,\log b_g)|(\log a_0,\log b_0),\Lambda^{-1}) \\
&= \sum_{g=1}^{G}\log\mathcal{N}((\log a_g,\log b_g)|(\log a_0,\log b_0),\Lambda^{-1}) \\
&= \sum_{g=1}^{G}\left(-\log\sqrt{(2\pi)^2|\Lambda^{-1}|}-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^{\mathrm{T}}\Lambda(\boldsymbol{x}-\boldsymbol{\mu})\right) \\
&\quad \text{with } \boldsymbol{x}=\begin{pmatrix}\log a_g\\ \log b_g\end{pmatrix},\boldsymbol{\mu}=\begin{pmatrix}\log a_0\\ \log b_0\end{pmatrix}, \\
&\quad \Lambda=\begin{pmatrix}\lambda_{aa} & \lambda_{ab}\\ \lambda_{ab} & \lambda_{bb}\end{pmatrix} \\
&= \sum_{g=1}^{G}\left(-2\log(2\pi)^{\frac{1}{2}}-\frac{1}{2}\log|\Lambda|^{-1}+-\frac{1}{2}(\bar{a}_g^2\lambda_{aa}+2\bar{a}_g\bar{b}_g\lambda_{ab}+\bar{b}_g^2\lambda_{bb})\right) \\
&\quad \text{with } \bar{a}_g=\log a_g-\log a_0=\log\frac{a_g}{a_0},\bar{b}_g=\log b_g-\log b_0=\log\frac{b_g}{b_0} \\
&\quad \text{and } |\Lambda|=\lambda_{aa}\lambda_{bb}\,(1-e^{-\kappa}),\text{ this leads to} \\
&= \sum_{g=1}^{G}[-\log(2\pi)+\frac{1}{2}\log\lambda_{aa}+\frac{1}{2}\log\lambda_{bb}+\frac{1}{2}\log(1-e^{-\kappa}) \\
&\quad -\frac{1}{2}\left(\log\frac{a_g}{a_0}\right)^2\lambda_{aa}-\log\frac{a_g}{a_0}\log\frac{b_g}{b_0}\lambda_{ab}-\frac{1}{2}\left(\log\frac{b_g}{b_0}\right)^2\lambda_{bb}] \qquad \text{(B.13)}
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial}{\partial a_0}\log p(\boldsymbol{a},\boldsymbol{b}|a_0,b_0,\Lambda^{-1}) &= -\sum_{g=1}^{G}\left(\lambda_{aa}\log\frac{a_g}{a_0}\left(-\frac{1}{a_0}\right)+\lambda_{ab}\log\frac{b_g}{b_0}\left(-\frac{1}{a_0}\right)\right) \\
&= \frac{1}{a_0}\sum_{g=1}^{G}(\lambda_{aa}\log\frac{a_g}{a_0}+\lambda_{ab}\log\frac{b_g}{b_0}) \qquad \text{(B.14)}
\end{aligned}
$$

$$
\boxed{\frac{\partial}{\partial a_0}\log p(\boldsymbol{a},\boldsymbol{b}|a_0,b_0,\Lambda^{-1})=\frac{1}{a_0}\sum_{g=1}^{G}(\lambda_{aa}\log\frac{a_g}{a_0}+\lambda_{ab}\log\frac{b_g}{b_0})} \qquad \text{(B.15)}
$$

$$
\boxed{\frac{\partial}{\partial b_0}\log p(\boldsymbol{a},\boldsymbol{b}|a_0,b_0,\Lambda^{-1})=\frac{1}{b_0}\sum_{g=1}^{G}(\lambda_{bb}\log\frac{b_g}{b_0}+\lambda_{ab}\log\frac{a_g}{a_0})} \qquad \text{(B.16)}
$$

$$
\begin{aligned}
\frac{\partial}{\partial a_g}\log p(\boldsymbol{a},\boldsymbol{b}|a_0,b_0,\Lambda^{-1}) &= -\frac{1}{2}\left(2\lambda_{aa}\log\frac{a_g}{a_0}\frac{1}{a_g}+2\lambda_{ab}\log\frac{b_g}{b_0}\frac{1}{a_g}\right) \\
&= -\frac{1}{a_g}(\lambda_{aa}\log\frac{a_g}{a_0}+\lambda_{ab}\log\frac{b_g}{b_0}) \qquad \text{(B.17)}
\end{aligned}
$$

$$\boxed{\frac{\partial}{\partial a_g}\log p(\boldsymbol{a},\boldsymbol{b}|a_0,b_0,\Lambda^{-1}) = -\frac{1}{a_g}(\lambda_{aa}\log\frac{a_g}{a_0}+\lambda_{ab}\log\frac{b_g}{b_0})} \tag{B.18}$$

$$\begin{aligned}\frac{\partial}{\partial b_g}\log p(\boldsymbol{a},\boldsymbol{b}|a_0,b_0,\Lambda^{-1}) &= -\frac{1}{2}\left(2\lambda_{bb}\log\frac{b_g}{b_0}\frac{1}{b_g}+2\lambda_{ab}\log\frac{a_g}{a_0}\frac{1}{b_g}\right)\\ &= -\frac{1}{b_g}(\lambda_{bb}\log\frac{b_g}{b_0}+\lambda_{ab}\log\frac{a_g}{a_0})\end{aligned} \tag{B.19}$$

$$\boxed{\frac{\partial}{\partial b_g}\log p(\boldsymbol{a},\boldsymbol{b}|a_0,b_0,\Lambda^{-1}) = -\frac{1}{b_g}(\lambda_{bb}\log\frac{b_g}{b_0}+\lambda_{ab}\log\frac{a_g}{a_0})} \tag{B.20}$$

$$\begin{aligned}\frac{\partial}{\partial \lambda_{aa}}\log p(\boldsymbol{a},\boldsymbol{b}|a_0,b_0,\Lambda^{-1}) &= \frac{\partial}{\partial\lambda_{aa}}\sum_{g=1}^{G}[-\log(2\pi)+\frac{1}{2}\log\lambda_{aa}+\frac{1}{2}\log\lambda_{bb}+\frac{1}{2}\log(1-e^{-\kappa})\\ &\quad \frac{1}{2}\bar{a}_g^2\lambda_{aa}-\bar{a}_g\bar{b}_g\lambda_{ab}-\frac{1}{2}\bar{b}_g^2\lambda_{bb}]\\ &\quad \text{and with } \lambda_{ab}=\sqrt{\lambda_{aa}}\sqrt{\lambda_{bb}}e^{-\frac{\kappa}{2}} \text{ (follows by eq. 4.11) :}\\ &= \sum_{g=1}^{G}\frac{1}{2\lambda_{aa}}-\frac{1}{2}\bar{a}_g^2-\frac{\partial}{\partial\lambda_{aa}}\bar{a}_g\bar{b}_g\sqrt{\lambda_{aa}}\sqrt{\lambda_{bb}}e^{-\frac{\kappa}{2}}\\ &= \sum_{g=1}^{G}\frac{1}{2\lambda_{aa}}-\frac{1}{2}\bar{a}_g^2-\frac{\bar{a}_g\bar{b}_g\sqrt{\lambda_{bb}}e^{-\frac{\kappa}{2}}}{2\sqrt{\lambda_{aa}}}\\ &= \frac{G}{2\lambda_{aa}}-\sum_{g=1}^{G}\frac{1}{2}\bar{a}_g^2-\frac{\bar{a}_g\bar{b}_g\sqrt{\lambda_{bb}}e^{-\frac{\kappa}{2}}}{2\sqrt{\lambda_{aa}}}\\ &= \frac{G}{2\lambda_{aa}}-\frac{1}{2}\sum_{g=1}^{G}\bar{a}_g^2-\frac{\sqrt{\lambda_{bb}}e^{-\frac{\kappa}{2}}}{2\sqrt{\lambda_{aa}}}\sum_{g=1}^{G}\bar{a}_g\bar{b}_g\end{aligned} \tag{B.21}$$

$$\boxed{\frac{\partial}{\partial \lambda_{aa}}\log p(\boldsymbol{a},\boldsymbol{b}|a_0,b_0,\Lambda^{-1}) = \frac{G}{2\lambda_{aa}}-\frac{1}{2}\sum_{g=1}^{G}\bar{a}_g^2-\frac{\sqrt{\lambda_{bb}}e^{-\frac{\kappa}{2}}}{2\sqrt{\lambda_{aa}}}\sum_{g=1}^{G}\bar{a}_g\bar{b}_g} \tag{B.22}$$

$$\begin{aligned}
\frac{\partial}{\partial \lambda_{bb}} \log p(\boldsymbol{a}, \boldsymbol{b}|a_0, b_0, \Lambda^{-1}) &= \frac{\partial}{\partial \lambda_{bb}} \sum_{g=1}^{G} [-\log(2\pi) + \frac{1}{2}\log\lambda_{aa} + \frac{1}{2}\log\lambda_{bb} + \frac{1}{2}\log(1-e^{-\kappa}) \\
&\quad -\frac{1}{2}\bar{a}_g^2\lambda_{aa} - \bar{a}_g\bar{b}_g\lambda_{ab} - \frac{1}{2}\bar{b}_g^2\lambda_{bb}] \\
&= \sum_{g=1}^{G} \frac{1}{2\lambda_{bb}} - -\frac{\partial}{\partial \lambda_{bb}} \bar{a}_g\bar{b}_g\sqrt{\lambda_{aa}}\sqrt{\lambda_{bb}}e^{-\frac{\kappa}{2}} - \frac{1}{2}\bar{b}_g^2 \\
&= \sum_{g=1}^{G} \frac{1}{2\lambda_{bb}} - \frac{\bar{a}_g\bar{b}_g\sqrt{\lambda_{aa}}e^{-\frac{\kappa}{2}}}{2\sqrt{\lambda_{bb}}} - \frac{1}{2}\bar{b}_g^2 \\
&= \frac{G}{2\lambda_{bb}} - \frac{1}{2}\sum_{g=1}^{G}\bar{b}_g^2 - \frac{\sqrt{\lambda_{aa}}e^{-\frac{\kappa}{2}}}{2\sqrt{\lambda_{bb}}}\sum_{g=1}^{G}\bar{a}_g\bar{b}_g \qquad \text{(B.23)}
\end{aligned}$$

$$\boxed{\frac{\partial}{\partial \lambda_{bb}} \log p(\boldsymbol{a}, \boldsymbol{b}|a_0, b_0, \Lambda^{-1}) = \frac{G}{2\lambda_{bb}} - \frac{1}{2}\sum_{g=1}^{G}\bar{b}_g^2 - \frac{\sqrt{\lambda_{aa}}e^{-\frac{\kappa}{2}}}{2\sqrt{\lambda_{bb}}}\sum_{g=1}^{G}\bar{a}_g\bar{b}_g} \qquad \text{(B.24)}$$

$$\begin{aligned}
\frac{\partial}{\partial \kappa} \log p(\boldsymbol{a}, \boldsymbol{b}|a_0, b_0, \Lambda^{-1}) &= \frac{\partial}{\partial \lambda_{aa}} \sum_{g=1}^{G} [-\log(2\pi) + \frac{1}{2}\log\lambda_{aa} + \frac{1}{2}\log\lambda_{bb} + \frac{1}{2}\log(1-e^{-\kappa}) \\
&\quad -\frac{1}{2}(\bar{a}_g^2\lambda_{aa} + 2\bar{a}_g\bar{b}_g\lambda_{ab} + \bar{b}_g^2\lambda_{bb})] \\
&= \sum_{g=1}^{G} \left(\frac{\partial}{\partial \kappa}\frac{1}{2}\log(1-e^{-k}) - \frac{\partial}{\partial \kappa}(\bar{a}_g\bar{b}_g\sqrt{\lambda_{aa}}\sqrt{\lambda_{bb}}e^{-\frac{\kappa}{2}}) \right) \\
&= \sum_{g=1}^{G} \left(\frac{1}{2}\frac{e^{-\kappa}}{(1-e^{-\kappa})} + \bar{a}_g\bar{b}_g\sqrt{\lambda_{aa}}\sqrt{\lambda_{bb}}\frac{1}{2}e^{-\frac{\kappa}{2}} \right) \\
&= \frac{G}{2}\frac{e^{-\kappa}}{(1-e^{-\kappa})} + \sum_{g=1}^{G}(\bar{a}_g\bar{b}_g\sqrt{\lambda_{aa}}\sqrt{\lambda_{bb}}\frac{1}{2}e^{-\frac{\kappa}{2}}) \\
&= \frac{G}{2}\frac{e^{-\kappa}}{(1-e^{-\kappa})} + \sqrt{\lambda_{aa}}\sqrt{\lambda_{bb}}\frac{1}{2}e^{-\frac{\kappa}{2}}\sum_{g=1}^{G}\bar{a}_g\bar{b}_g \qquad \text{(B.25)}
\end{aligned}$$

$$\boxed{\frac{\partial}{\partial \kappa} \log p(\boldsymbol{a}, \boldsymbol{b}|a_0, b_0, \Lambda^{-1}) = \frac{G}{2}\frac{e^{-\kappa}}{(1-e^{-\kappa})} + \sqrt{\lambda_{aa}}\sqrt{\lambda_{bb}}\frac{1}{2}e^{-\frac{\kappa}{2}}\sum_{g=1}^{G}\bar{a}_g\bar{b}_g} \qquad \text{(B.26)}$$

$$\begin{aligned}
\log p(\boldsymbol{c}|\sigma_c^2) &= \log \prod_{n=1}^{N} \mathcal{N}(c_n|0,\sigma_c^2) \\
&= \sum_{n=1}^{N} \log \mathcal{N}(c_n|0,\sigma_c^2) \\
&= \sum_{n=1}^{N} \log \left(\frac{1}{\sqrt{2\pi\sigma_c^2}} \exp\left(-\frac{c_n^2}{2\sigma_c^2}\right)\right) \\
&= \sum_{n=1}^{N} \left(-\log\sqrt{2\pi} - \log\sigma_c - \frac{c_n^2}{2\sigma_c^2}\right) \\
&= -\sum_{n=1}^{N} \left(\log\sqrt{2\pi} + \log\sigma_c + \frac{c_n^2}{2\sigma_c^2}\right) \qquad \text{(B.27)}
\end{aligned}$$

$$\boxed{\frac{\partial}{\partial c_n} \log p(\boldsymbol{c}|\sigma_c^2) = -\frac{c_n}{\sigma_c^2}} \qquad \text{(B.28)}$$

$$\begin{aligned}
\frac{\partial}{\partial \sigma_c} \log p(\boldsymbol{c}|\sigma_c^2) &= \sum_{n=1}^{N} \frac{\partial}{\partial \sigma_c} \left(-\log\sqrt{2\pi} - \log\sigma_c - \frac{c_n^2}{2\sigma_c^2}\right) \\
&= \sum_{n=1}^{N} -\frac{1}{\sigma_c} - \frac{c_n^2}{2}\frac{(-1)}{\sigma_c^4} 2\sigma_c \\
&= \sum_{n=1}^{N} -\frac{1}{\sigma_c} + \frac{c_n^2}{\sigma_c^3} \\
&= \sum_{n=1}^{N} \frac{c_n^2 - \sigma_c^2}{\sigma_c^3} \qquad \text{(B.29)}
\end{aligned}$$

$$\boxed{\frac{\partial}{\partial \sigma_c} \log p(\boldsymbol{c}|\sigma_c^2) = \sum_{n=1}^{N} \frac{c_n^2 - \sigma_c^2}{\sigma_c^3}} \qquad \text{(B.30)}$$

- for $p(\boldsymbol{X}|\theta)$

  Again, it is not necessary to calculate *everything* since not all variables appear. The partial derivatives that are needed are the ones with respect to $\rho_k, c_n, \mu_{kg}, a_g, b_g$.

  In the calculations the following notation is used to improve readability and make the calculations easier:

  $$\begin{aligned} m_{kg} &= s_n \mu_{kg} \\ \sigma_{kg}^2 &= c_n(a_g s_n^2 \mu_{kg}^2 + b_g s_n \mu_{kg}) \end{aligned}$$

  furthermore we define $p_{kg} = \frac{\sigma_{kg}^2 - m_{kg}}{\sigma_{kg}^2}$ and $r = \frac{m_{kg}^2}{\sigma_{kg}^2 - m_{kg}}$, p_{kg} being the probability of success and r_{kg} the predefined number of failures, both parametrized by the mean and variance of the distribution.

$$
\begin{aligned}
p(\mathbf{X}|\boldsymbol{\mu},\boldsymbol{\rho},\boldsymbol{a},\boldsymbol{b},\boldsymbol{c}) &= \prod_{n=1}^{N}\sum_{k=1}^{K} p(k)\, p(\mathbf{x}_n|\boldsymbol{\mu}_k,\boldsymbol{s},\boldsymbol{a},\boldsymbol{b}) \\
&= \prod_{n=1}^{N}\sum_{k=1}^{K} \frac{e^{\rho_k}}{\sum_{k'} e^{\rho_{k'}}} \prod_{g=1}^{G} \mathrm{NegBin}(x_{ng}|\mu = s_n\mu_{kg}, \sigma^2 = c_n(a_g\mu^2 + b_g\mu)) \\
p(\boldsymbol{X}|\boldsymbol{\theta}) &= \prod_{n=1}^{N}\sum_{k=1}^{K} \frac{e^{\rho_k}}{\sum_{k'} e^{\rho_{k'}}} \prod_{g=1}^{G} \mathrm{NegBin}(x_{ng}|r_{kg}, p_{kg}) \qquad \text{(B.31)}
\end{aligned}
$$

$$
\begin{aligned}
\log p(\boldsymbol{X}|\boldsymbol{\theta}) &= \log \prod_{n=1}^{N}\sum_{k=1}^{K} p(k)\, p(\mathbf{x}_n|\boldsymbol{\mu}_k,\boldsymbol{s},\boldsymbol{a},\boldsymbol{b}) \\
&= \sum_{n=1}^{N} \log \sum_{k=1}^{K} p(k)\, p(\mathbf{x}_n|\boldsymbol{\mu}_k) \qquad \text{(B.32)}
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial}{\partial \rho_k}\log p(\boldsymbol{X}|\boldsymbol{\theta}) &= \sum_{n=1}^{N} \log \sum_{k'=1}^{K} p(k')\, p(\mathbf{x}_n|\boldsymbol{\mu}_{k'}) \\
&= \sum_{n=1}^{N} \frac{1}{\sum_{k'=1}^{K} p(k')\, p(\mathbf{x}_n|\boldsymbol{\mu}_{k'})} \frac{\partial}{\partial \rho_k} \sum_{k'=1}^{K} p(k')\, p(\mathbf{x}_n|\boldsymbol{\mu}_{k'}) \\
&= \sum_{n=1}^{N} \frac{1}{p(\mathbf{x}_n|\boldsymbol{\theta})} \sum_{k'=1}^{K} \frac{\partial}{\partial \rho_k} p(k)\, p(\mathbf{x}_n|\boldsymbol{\mu}_{k'}) \\
&= \sum_{n=1}^{N} \frac{1}{p(\mathbf{x}_n|\boldsymbol{\theta})} \sum_{k'=1}^{K} p(\mathbf{x}_n|\boldsymbol{\mu}_{k'}) \frac{\partial}{\partial \rho_k} \frac{e^{\rho_{k'}}}{\sum_{k'} e^{\rho_{k'}}} \\
&= \sum_{n=1}^{N} \frac{1}{p(\mathbf{x}_n|\boldsymbol{\theta})} \sum_{k'=1}^{K} p(\mathbf{x}_n|\boldsymbol{\mu}_k) \frac{\frac{\partial}{\partial \rho_k} e^{\rho_{k'}} \sum_{i=1}^{K} e^{\rho_i} - e^{\rho_{k'}} \frac{\partial}{\partial \rho_k} \sum_{i=1}^{K} e^{\rho_i}}{(\sum_{k'} e^{\rho_{k'}})^2} \\
&\quad \frac{\partial}{\partial \rho_k} e^{\rho_{k'}} = 0 \text{ for } k' \neq k, e^{\rho_k} \text{ else} \\
&= \sum_{n=1}^{N} \frac{1}{p(\mathbf{x}_n|\boldsymbol{\theta})} \left(\sum_{k'=1}^{K} p(\mathbf{x}_n|\boldsymbol{\mu}_k) \frac{-e^{\rho_{k'}} e^{\rho_k}}{(\sum_{k'} e^{\rho_{k'}})^2} \right) + \frac{e^{\rho_k} \sum_{i=1}^{K} e^{\rho_i}}{(\sum_{i=1}^{K} e^{\rho_i})^2} p(\mathbf{x}_n|\boldsymbol{\mu}_k) \\
&= \sum_{n=1}^{N} \frac{1}{p(\mathbf{x}_n|\boldsymbol{\theta})} \left[\left(\sum_{k'=1}^{K} p(\mathbf{x}_n|\boldsymbol{\mu}_k) \frac{-e^{\rho_{k'}}}{\sum_{k'} e^{\rho_{k'}}} \frac{e^{\rho_k}}{\sum_{k'} e^{\rho_{k'}}} \right) + \frac{e^{\rho_k}}{\sum_{i=1}^{K} e^{\rho_i}} p(\mathbf{x}_n|\boldsymbol{\mu}_k) \right] \\
&= \sum_{n=1}^{N} \frac{1}{p(\mathbf{x}_n|\boldsymbol{\theta})} \left(p(k)p(\mathbf{x}_n|\boldsymbol{\mu}_k) - \sum_{k'=1}^{K} p(k)p(k')p(\mathbf{x}_n|\boldsymbol{\mu}_k) \right) \qquad \text{(B.33)}
\end{aligned}
$$

$$
\boxed{\frac{\partial}{\partial \rho_k}\log p(\boldsymbol{X}|\boldsymbol{\theta}) = \sum_{n=1}^{N} \frac{1}{p(\mathbf{x}_n|\boldsymbol{\theta})} \left(p(k)p(\mathbf{x}_n|\boldsymbol{\mu}_k) - \sum_{k'=1}^{K} p(k)p(k')p(\mathbf{x}_n|\boldsymbol{\mu}_k) \right)} \qquad \text{(B.34)}
$$

in the case of all other variables of the expression we are interested in the $p(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{s})$ term which contains the relevant variables.

$$\begin{aligned}\log p(\boldsymbol{X}|\boldsymbol{\theta}) &= \log \prod_{n=1}^{N} \sum_{k=1}^{K} p(k)\, p(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{s}, \boldsymbol{a}, \boldsymbol{b}) \\ &= \sum_{n=1}^{N} \log \sum_{k=1}^{K} p(k)\, p(\mathbf{x}_n|\boldsymbol{\mu}_k)\end{aligned} \tag{B.35}$$

$$\begin{aligned}\frac{\partial}{\partial c_n} \log p(\boldsymbol{X}|\boldsymbol{\theta}) &= \frac{1}{\sum_{k=1}^{K} p(k)\, p(\mathbf{x}_n|\boldsymbol{\mu}_k)} \frac{\partial}{\partial c_n} \sum_{k=1}^{K} p(k)\, p(\mathbf{x}_n|\boldsymbol{\mu}_k) \\ &= \frac{1}{p(\boldsymbol{x}_n|\boldsymbol{\theta})} \sum_{k=1}^{K} \left(p(k) \frac{\partial}{\partial c_n} p(\mathbf{x}_n|\boldsymbol{\mu}_k) \right) \\ &= \frac{1}{p(\boldsymbol{x}_n|\boldsymbol{\theta})} \sum_{k=1}^{K} \left(p(k) \frac{\partial}{\partial c_n} \prod_{g=1}^{G} \mathrm{NegBin}(x_{ng}|r_{kg}, p_{kg}) \right) \\ &= \frac{1}{p(\boldsymbol{x}_n|\boldsymbol{\theta})} \sum_{k=1}^{K} \left(p(k) \left[\prod_{g=1}^{G} \mathrm{NegBin}(x_{ng}|r_{kg}, p_{kg}) \right] \frac{\partial}{\partial c_n} \log \prod_{g=1}^{G} \mathrm{NegBin}(x_{ng}|r_{kg}, p_{kg}) \right) \\ &= \frac{1}{p(\boldsymbol{x}_n|\boldsymbol{\theta})} \sum_{k=1}^{K} \left(p(k) \left[\prod_{g=1}^{G} \mathrm{NegBin}(x_{ng}|r_{kg}, p_{kg}) \right] \sum_{g=1}^{G} \frac{\partial}{\partial c_n} \log \mathrm{NegBin}(x_{ng}|r_{kg}, p_{kg}) \right)\end{aligned} \tag{B.36}$$

we only need to compute $\frac{\partial}{\partial c_n} \log \mathrm{NegBin}(x_{ni}|r_{kg}, p_{kg})$ and we are only interested in terms that contain the variance σ_{kg}^2, since only those contain c_n.

$$\begin{aligned}\log \mathrm{NegBin}(x_{ng}|r_{ng}, p_{kg}) &= \log \left(\frac{\Gamma(x_{ng} + r_{ng})}{x_{ng}!\Gamma(r)} (1 - p_{kg})^r p_{kg}^{x_{ng}} \right) \\ &= \log \Gamma(x_{ng} + r_{kg}) - \log(x_{ng}!) - \log \Gamma(r_{kg}) \\ &\quad + r_{kg} \log(1 - p_{kg}) + x_{ng} \log(p_{kg})\end{aligned} \tag{B.37}$$

before going on with the calculations we need to calculate the derivatives of $1 - p_{kg}$ and r_{kg}, since we will need them a lot. In this case we can ignore m_{kg} as it doesn't contain c_n.

$$\begin{aligned}
\frac{\partial r_{kg}}{\partial c_n} &= \frac{\partial}{\partial c_n}\frac{m_{kg}^2}{\sigma_{kg}^2 - m_{kg}} \\
&= -\frac{m_{kg}^2}{(\sigma_{kg}^2 - m_{kg})^2}\sigma_{kg}^{2\prime} \\
&= -\frac{m_{kg}^2}{(\sigma_{kg}^2 - m_{kg})^2}(a_g s_n^2 \mu_{kg}^2 + b_g s_n^2 \mu_{kg}) \\
&= -\frac{m_{kg}^2}{(\sigma_{kg}^2 - m_{kg})^2}\frac{\sigma_{kg}^2}{c_n} \\
&= -\frac{m_{kg}^2}{(\sigma_{kg}^2 - m_{kg})}\frac{\sigma_{kg}^2}{(\sigma_{kg}^2 - m_{kg})}\frac{1}{c_n} \\
&= -\frac{r_{kg}}{p_{kg}}\frac{1}{c_n}
\end{aligned} \tag{B.38}$$

$$\begin{aligned}
\frac{\partial(1 - p_{kg})}{\partial c_n} &= \frac{\partial}{\partial c_n}1 - \frac{\sigma_{kg}^2 - m_{kg}}{\sigma_{kg}^2} \\
&= \frac{\partial}{\partial c_n}\frac{m_{kg}}{\sigma_{kg}^2} \\
&= -\frac{m_{kg}}{\sigma_{kg}^4}\sigma_{kg}^{2\prime} \\
&= -\frac{m_{kg}}{\sigma_{kg}^4}(a_g s_n^2 \mu_{kg}^2 + b_g s_n \mu_{kg}) \\
&= -\frac{m_{kg}}{\sigma_{kg}^2}\frac{1}{\sigma_{kg}^2}\frac{\sigma_{kg}^2}{c_n} \\
&= \frac{p_{kg} - 1}{c_n}
\end{aligned} \tag{B.39}$$

$$\begin{aligned}
\frac{\partial p_{kg}}{\partial c_n} &= -\frac{\partial r_{kg}}{\partial c_n} \\
&= \frac{1 - p_{kg}}{c_n}
\end{aligned} \tag{B.40}$$

and now continuing from eq. B.37:

$$
\begin{aligned}
\frac{\partial}{\partial c_n}\log \mathrm{NegBin}(x_{ng}|r_{ng},p_{kg}) &= \frac{1}{\Gamma(x_{ng}+r_{kg})}\Gamma(x_{ng}+r_{kg})\psi^0(x_{ng}+r_{kg})\frac{\partial}{\partial c_n}r_{kg} \\
&\quad -\frac{1}{\Gamma(r_{kg})}\Gamma(r_{kg})\psi^0(r_{kg})\frac{\partial}{\partial c_n}r_{kg} \\
&\quad +\log(1-p_{kg})\frac{\partial}{\partial c_n}r_{kg}+r_{kg}\frac{1}{(1-p_{kg})}\frac{\partial}{\partial c_n}(1-p_{kg})+x_{ng}\frac{1}{p_{kg}}\frac{\partial}{\partial c_n}p_{kg} \\
&= -\psi^0(x_{ng}+r_{kg})\frac{r_{kg}}{p_{kg}}\frac{1}{c_n}+\psi^0(r_{kg})\frac{r_{kg}}{p_{kg}}\frac{1}{c_n}+\log(1-p_{kg})\frac{r_{kg}}{p_{kg}}\frac{1}{c_n} \\
&\quad r_{kg}\frac{1}{(1-p_{kg})}\frac{p_{kg}-1}{c_n}+x_{ng}\frac{1}{p_{kg}}\frac{1-p_{kg}}{c_n} \\
&= \frac{1}{c_n}\left(\frac{r_{kg}}{p_{kg}}(\psi^0(r_{kg})-\psi^0(x_{ng}+r_{kg})+\log(1-p_{kg}))\right. \\
&\quad \left.-r_{kg}+x_{ng}\frac{1-p_{kg}}{p_{kg}}\right) \qquad \text{(B.41)}
\end{aligned}
$$

The same procedure can be followed when deriving by s_n, as it only occurs inside the product. The general form of the derivative is the same (equations B.36 and B.37). We recalculate ∂r and $\partial(1-p_{kg})$, this time following the terms that contain m_{kg}.

$$
\begin{aligned}
\frac{\partial r_{kg}}{\partial s_n} &= \frac{\partial}{\partial s_n}\frac{m_{kg}^2}{\sigma_{kg}^2 - m_{kg}} \\
&= \frac{m_{kg}^{2\prime}(\sigma_{kg}^2 - m_{kg}) - (\sigma_{kg}^2 - m_{kg})' m_{kg}^2}{(\sigma_{kg}^2 - m_{kg})^2} \\
&= \frac{2 s_n \mu_{kg}^2 (\sigma_{kg}^2 - m_{kg}) - s_n^2 \mu_{kg}^2 (c_n a_g s_n^2 \mu_{kg}^2 + c_n b_g s_n \mu_{kg} - s_n \mu_{kg})'}{(\sigma_{kg}^2 - m_{kg})^2} \\
&= \frac{2 s_n \mu_{kg}^2 (\sigma_{kg}^2 - m_{kg}) - s_n^2 \mu_{kg}^2 (2 c_n a_g s_n \mu_{kg}^2 + c_n b_g \mu_{kg} - \mu_{kg})}{(\sigma_{kg}^2 - m_{kg})^2} \\
&= \frac{2 s_n \mu_{kg}^2 (\sigma_{kg}^2 - m_{kg}) - s_n \mu_{kg}^2 (c_n a_g s_n^2 \mu_{kg}^2 + c_n b_g s_n \mu_{kg} - s_n \mu_{kg} + c_n a_g s_n^2 \mu_{kg}^2)}{(\sigma_{kg}^2 - m_{kg})^2} \\
&= \frac{2 s_n \mu_{kg}^2 (\sigma_{kg}^2 - m_{kg}) - s_n \mu_{kg}^2 (\sigma_{kg}^2 - m_{kg} + c_n a_g s_n^2 \mu_{kg}^2)}{(\sigma_{kg}^2 - m_{kg})^2} \\
&= \frac{s_n \mu_{kg}^2 (\sigma_{kg}^2 - m_{kg} - c_n a_g s_n^2 \mu_{kg}^2)}{(\sigma_{kg}^2 - m_{kg})^2} \\
&= \frac{s_n \mu_{kg}^2 (c_n a_g s_n^2 \mu_{kg}^2 + c_n b_g s_n \mu_{kg} - s_n \mu_{kg} - c_n a_g s_n^2 \mu_{kg}^2)}{(\sigma_{kg}^2 - m_{kg})^2} \\
&= \frac{s_n \mu_{kg}^2 (c_n b_g s_n \mu_{kg} - s_n \mu_{kg})}{(\sigma_{kg}^2 - m_{kg})^2} \\
&= \frac{s_n^2 \mu_{kg}^3 (c_n b_g - 1)}{(\sigma_{kg}^2 - m_{kg})^2} \\
&= \frac{s_n^2 \mu_{kg}^2}{\sigma_{kg}^2 - m_{kg}} \frac{\mu_{kg}(c_n b_g - 1)}{\sigma_{kg}^2 - m_{kg}} \\
&= r_{kg} \frac{\mu_{kg}(c_n b_g - 1)}{\sigma_{kg}^2 - m_{kg}}
\end{aligned}
\tag{B.42}
$$

$$\begin{aligned}
\frac{\partial(1-p_{kg})}{\partial s_n} &= \frac{\partial}{\partial s_n}\frac{m_{kg}}{\sigma^2_{kg}} \\
&= \frac{m'_{kg}\sigma^2_{kg} - m_{kg}\sigma^{2\prime}_{kg}}{\sigma^4_{kg}} \\
&= \frac{\mu_{kg}\sigma^2_{kg} - s_n\mu_{kg}(2c_n a_g s_n \mu^2_{kg} + c_n b_g \mu_{kg})}{\sigma^4_{kg}} \\
&= \frac{\mu_{kg}\sigma^2_{kg} - \mu_{kg}(\sigma^2_{kg} + c_n a_g s_n \mu^2_{kg})}{\sigma^4_{kg}} \\
&= -\frac{c_n a_g s^2_n \mu^3_{kg}}{\sigma^4_{kg}} \\
&= -\frac{s^2_n \mu^2_{kg}}{\sigma^4_{kg}} c_n a_g \mu_{kg} \\
&= -(1-p_{kg})^2 c_n a_g \mu_{kg}
\end{aligned} \tag{B.43}$$

$$\begin{aligned}
\frac{\partial p_{kg}}{\partial s_n} &= -\frac{\partial}{\partial s_n}(1-p_{kg}) \\
&= (1-p_{kg})^2 c_n a_g \mu_{kg}
\end{aligned} \tag{B.44}$$

and now continuing from B.37:

$$\begin{aligned}
\frac{\partial}{\partial s_n}\log \mathrm{NegBin}(x_{ng}|r_{ng},p_{kg}) &= \frac{1}{\Gamma(x_{ng}+r_{kg})}\Gamma(x_{ng}+r_{kg})\psi^0(x_{ng}+r_{kg})\frac{\partial}{\partial s_n}r_{kg} \\
&\quad -\frac{1}{\Gamma(r_{kg})}\Gamma(r_{kg})\psi^0(r_{kg})\frac{\partial}{\partial s_n}r_{kg} + \log(1-p_{kg})\frac{\partial}{\partial s_n}r_{kg} \\
&\quad +r_{kg}\frac{1}{(1-p_{kg})}\frac{\partial}{\partial s_n}(1-p_{kg}) + x_{ng}\frac{1}{p_{kg}}\frac{\partial}{\partial s_n}p_{kg} \\
&= \psi^0(x_{ng}+r_{kg})\frac{\partial r_{kg}}{\partial s_n} - \psi^0(r_{kg})\frac{\partial r_{kg}}{\partial s_n} + \log(1-p_{kg})\frac{\partial r_{kg}}{\partial s_n} \\
&\quad -\frac{r_{kg}}{(1-p_{kg})}(1-p_{kg})^2 c_n a_g \mu_{kg} + \frac{x_{ng}}{p_{kg}}(1-p_{kg})^2 c_n a_g \mu_{kg} \\
&= \frac{\partial r_{kg}}{\partial s_n}(\psi^0(x_{ng}+r_{kg}) - \psi^0(r_{kg}) + \log(1-p_{kg})) \\
&\quad +\left(\frac{x_{ng}}{p_{kg}} - \frac{r_{kg}}{(1-p_{kg})}\right)(1-p_{kg})^2 c_n a_g \mu_{kg}
\end{aligned} \tag{B.45}$$

For the derivatives with respect to a_g, b_g and μ_{kg} the overall structure is the same. Here we get to keep the sum over all cells (n) but since the derivatives for $g' \neq g$ are 0 we do not need the product rule:

$$
\begin{aligned}
\frac{\partial}{\partial a_g}\log p(\boldsymbol{X}|\boldsymbol{\theta}) &= \frac{\partial}{\partial a_g}\log\prod_{n=1}^{N}\sum_{k=1}^{K}p(k)\,p(\mathbf{x}_n|\boldsymbol{\mu}_k,\boldsymbol{s},\boldsymbol{a},\boldsymbol{b}) \\
&= \frac{\partial}{\partial a_g}\sum_{n=1}^{N}\log\sum_{k=1}^{K}p(k)\,p(\mathbf{x}_n|\boldsymbol{\mu}_k) \\
&= \sum_{n=1}^{N}\left(\frac{1}{\sum_{k=1}^{K}p(k)\,p(\mathbf{x}_n|\boldsymbol{\mu}_k)}\frac{\partial}{\partial a_g}\sum_{k=1}^{K}(p(k)\,p(\mathbf{x}_n|\boldsymbol{\mu}_k))\right) \\
&= \sum_{n=1}^{N}\left(\frac{1}{p(\boldsymbol{x}_n|\boldsymbol{\theta})}\sum_{k=1}^{K}\left(p(k)\frac{\partial}{\partial a_g}p(\mathbf{x}_n|\boldsymbol{\mu}_k)\right)\right) \\
&= \sum_{n=1}^{N}\left(\frac{1}{p(\boldsymbol{x}_n|\boldsymbol{\theta})}\sum_{k=1}^{K}\left(p(k)\frac{\partial}{\partial a_g}\prod_{g'=1}^{G}p(\boldsymbol{x}_{ng'}|\boldsymbol{\theta})\right)\right) \\
&= \sum_{n=1}^{N}\left(\frac{1}{p(\boldsymbol{x}_n|\boldsymbol{\theta})}\sum_{k=1}^{K}\left(p(k)\left(\prod_{g'=1}^{G}p(\boldsymbol{x}_{ng'}|\boldsymbol{\theta})\right)\frac{\partial}{\partial a_g}\log p(\boldsymbol{x}_{ng}|\boldsymbol{\theta})\right)\right) \\
&= \sum_{n=1}^{N}\left(\frac{1}{p(\boldsymbol{x}_n|\boldsymbol{\theta})}\sum_{k=1}^{K}\left(p(k)\left(\prod_{g'=1}^{G}p(\boldsymbol{x}_{ng'}|\boldsymbol{\theta})\right)\frac{\partial}{\partial a_g}\log\mathrm{NegBin}(x_{ng}|r_{kg},p_{kg})\right)\right)
\end{aligned}
\tag{B.46}
$$

For the rest of the calculations we proceed in a similar matter; we need the derivatives of $1-p_{kg}$, r_{kg}. In the case of a_g (and later b_g) we are only interested in terms containing σ_{kg}^2.

$$
\begin{aligned}
\frac{\partial r_{kg}}{\partial a_g} &= \frac{\partial}{\partial a_g}\frac{m_{kg}^2}{\sigma_{kg}^2-m_{kg}} \\
&= -\frac{m_{kg}^2}{(\sigma_{kg}^2-m_{kg})^2}\sigma_{kg}^{2\prime} \\
&= -\frac{m_{kg}^2}{(\sigma_{kg}^2-m_{kg})^2}c_n s_n^2\mu_{kg}^2 \\
&= -r_{kg}\frac{c_n s_n^2\mu_{kg}^2}{\sigma_{kg}^2-m_{kg}}
\end{aligned}
\tag{B.47}
$$

$$\begin{aligned}
\frac{\partial(1-p_{kg})}{\partial a_g} &= \frac{\partial}{\partial a_g}1 - \frac{\sigma_{kg}^2 - m_{kg}}{\sigma_{kg}^2} \\
&= \frac{\partial}{\partial a_g}\frac{m_{kg}}{\sigma_{kg}^2} \\
&= -\frac{m_{kg}}{\sigma_{kg}^4}\sigma_{kg}^{2\prime} \\
&= -\frac{m_{kg}}{\sigma_{kg}^4}c_n s_n^2 \mu_{kg}^2 \\
&= -(1-p_{kg})\frac{c_n s_n^2 \mu_{kg}^2}{\sigma_{kg}^2}
\end{aligned} \tag{B.48}$$

$$\begin{aligned}
\frac{\partial p_{kg}}{\partial a_g} &= -\frac{\partial}{\partial a_g}(1-p_{kg}) \\
&= (1-p_{kg})\frac{c_n s_n^2 \mu_{kg}^2}{\sigma_{kg}^2}
\end{aligned} \tag{B.49}$$

and now continuing from B.37:

$$\begin{aligned}
\frac{\partial}{\partial a_g}\log \mathrm{NegBin}(x_{ng}|r_{ng},p_{kg}) &= \frac{1}{\Gamma(x_{ng}+r_{kg})}\Gamma(x_{ng}+r_{kg})\psi^0(x_{ng}+r_{kg})\frac{\partial}{\partial a_g}r_{kg} \\
&\quad -\frac{1}{\Gamma(r_{kg})}\Gamma(r_{kg})\psi^0(r_{kg})\frac{\partial}{\partial a_g}r_{kg} + \log(1-p_{kg})\frac{\partial}{\partial a_g}r_{kg} \\
&\quad +r_{kg}\frac{1}{(1-p_{kg})}\frac{\partial}{\partial a_g}(1-p_{kg}) + x_{ng}\frac{1}{p_{kg}}\frac{\partial}{\partial a_g}p_{kg} \\
&= \psi^0(x_{ng}+r_{kg})\frac{\partial r_{kg}}{\partial a_g} - \psi^0(r_{kg})\frac{\partial r_{kg}}{\partial a_g} + \log(1-p_{kg})\frac{\partial r_{kg}}{\partial a_g} \\
&\quad -\frac{r_{kg}}{(1-p_{kg})}\frac{\partial}{\partial a_g}p_{kg} + \frac{x_{ng}}{(1-p_{kg})}\frac{\partial}{\partial a_g}p_{kg} \\
&= \frac{\partial r_{kg}}{\partial a_g}(\psi^0(x_{ng}+r_{kg}) - \psi^0(r_{kg}) + \log(1-p_{kg})) \\
&\quad +\left(\frac{x_{ng}}{p_{kg}} - \frac{r_{kg}}{1-p_{kg}}\right)\frac{\partial p_{kg}}{\partial a_g}
\end{aligned} \tag{B.50}$$

The partial derivative with respect to b_g is virtually identical, with the exception of using μ_{kg} instead of μ_{kg}^2.

$$\frac{\partial}{\partial b_g}\log p(\boldsymbol{X}|\boldsymbol{\theta}) = \sum_{n=1}^{N}\left(\frac{1}{p(\boldsymbol{x}_n|\boldsymbol{\theta})}\sum_{k=1}^{K}\left(p(k)\prod_{g'=1}^{G}\left(p(\boldsymbol{x}_{ng'}|\boldsymbol{\theta})\frac{\partial}{\partial b_g}\log \mathrm{NegBin}(x_{ng}|r_{kg},p_{kg})\right)\right)\right) \tag{B.51}$$

$$\frac{\partial r_{kg}}{\partial b_g} = -r_{kg}\frac{c_n s_n \mu_{kg}}{\sigma_{kg}^2 - m_{kg}} \tag{B.52}$$

$$\frac{\partial p_{kg}}{\partial b_g} = (1-p_{kg})\frac{c_n s_n \mu_{kg}}{\sigma_{kg}^2} \tag{B.53}$$

$$\frac{\partial}{\partial b_g}\log \text{NegBin} = \frac{\partial r_{kg}}{\partial b_g}(\psi^0(x_{ng}+r_{kg}) - \psi^0(r_{kg}) + \log(1-p_{kg})) + \left(\frac{x_{ng}}{p_{kg}} - \frac{r_{kg}}{1-p_{kg}}\right)\frac{\partial p_{kg}}{\partial b_g} \tag{B.54}$$

For $\frac{\partial}{\partial \mu_{kg}}$ we have to keep all n's but we can throw away the k's we do not see. We also lose the g's we do not see.

$$\begin{aligned}
\frac{\partial}{\partial \mu_{kg}}\log p(\boldsymbol{X}|\boldsymbol{\theta}) &= \frac{\partial}{\partial \mu_{kg}}\log \prod_{n=1}^{N}\sum_{k=1}^{K} p(k)\, p(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{s}, \boldsymbol{a}, \boldsymbol{b}) \\
&= \sum_{n=1}^{N}\frac{\partial}{\partial \mu_{kg}}\log \sum_{k=1}^{K} p(k)\, p(\mathbf{x}_n|\boldsymbol{\mu}_k) \\
&= \sum_{n=1}^{N}\frac{1}{\sum_{k=1}^{K} p(k)\, p(\mathbf{x}_n|\boldsymbol{\mu}_k)}\frac{\partial}{\partial \mu_{kg}}\sum_{k=1}^{K} p(k)\, p(\mathbf{x}_n|\boldsymbol{\mu}_k) \\
&= \sum_{n=1}^{N}\left(\frac{1}{p(\boldsymbol{x}_n|\boldsymbol{\theta})}p(k)\frac{\partial}{\partial \mu_{kg}}p(\mathbf{x}_n|\boldsymbol{\mu}_k)\right) \\
&= \sum_{n=1}^{N}\left(\frac{1}{p(\boldsymbol{x}_n|\boldsymbol{\theta})}p(k)\left(\prod_{g'=1}^{G} p(\mathbf{x}_{ng'}|\boldsymbol{\mu}_k)\right)\frac{\partial}{\partial \mu_{kg}}\log p(\mathbf{x}_{ng}|\boldsymbol{\mu}_k)\right) \\
&= \sum_{n=1}^{N}\left(\frac{1}{p(\boldsymbol{x}_n|\boldsymbol{\theta})}p(k)\left(\prod_{g'=1}^{G} p(\mathbf{x}_{ng'}|\boldsymbol{\mu}_k)\right)\frac{\partial}{\partial \mu_{kg}}\log \text{NegBin}(x_{ng}|r, p_{kg})\right)
\end{aligned} \tag{B.55}$$

other than that we proceed in the same fashion, and calculate the helper derivatives:

$$
\begin{aligned}
\frac{\partial}{\partial \mu_{kg}} r_{kg} &= \frac{\partial}{\partial \mu_{kg}} \frac{m_{kg}^2}{\sigma_{kg}^2 - m_{kg}} \\
&= \frac{(m_{kg}^2)'(\sigma_{kg}^2 - m_{kg}) - m_{kg}^2 - (\sigma_{kg}^2 - m_{kg})'}{(\sigma_{kg}^2 - m_{kg})^2} \\
&= \frac{2\mu_{kg} s_n^2 (\sigma_{kg}^2 - m_{kg}) - (2 s_n^2 \mu_{kg} c_n a_g + b_g s_n c_n - s_n) s_n^2 \mu_{kg}^2}{(\sigma_{kg}^2 - m_{kg})^2} \\
&= \frac{2\mu_{kg} s_n^2 (\sigma_{kg}^2 - m_{kg}) - (2 s_n^2 \mu_{kg}^2 c_n a_g + s_n \mu_{kg} b_g c_n - s_n \mu_{kg}) s_n^2 \mu_{kg}}{(\sigma_{kg}^2 - m_{kg})^2} \\
&= \frac{2\mu_{kg} s_n^2 (\sigma_{kg}^2 - m_{kg}) - ((s_n^2 \mu_{kg}^2 c_n a_g + s_n \mu_{kg} b_g c_n) - s_n \mu_{kg} + s_n^2 \mu_{kg}^2 c_n a_g) s_n^2 \mu_{kg}}{(\sigma_{kg}^2 - m_{kg})^2} \\
&= \frac{2\mu_{kg} s_n^2 (\sigma_{kg}^2 - m_{kg}) - (\sigma_{kg}^2 - m_{kg} + s_n^2 \mu_{kg}^2 c_n a_g) s_n^2 \mu_{kg}}{(\sigma_{kg}^2 - m_{kg})^2} \\
&= \frac{\mu_{kg} s_n^2 (2(\sigma_{kg}^2 - m_{kg}) - (\sigma_{kg}^2 - m_{kg}) - s_n^2 \mu_{kg}^2 c_n a_g)}{(\sigma_{kg}^2 - m_{kg})^2} \\
&= \frac{\mu_{kg} s_n^2 (\sigma_{kg}^2 - m_{kg} - s_n^2 \mu_{kg}^2 c_n a_g)}{(\sigma_{kg}^2 - m_{kg})^2} \\
&= \frac{\mu_{kg} s_n^2 (s_n^2 \mu_{kg}^2 c_n a_g + \mu_{kg} c_n s_n b_g - s_n \mu_{kg} - s_n^2 \mu_{kg}^2 c_n a_g)}{(\sigma_{kg}^2 - m_{kg})^2} \\
&= \frac{\mu_{kg}^2 s_n^3 (c_n b_g - 1)}{(\sigma_{kg}^2 - m_{kg})^2} \\
&= \frac{m_{kg}^2}{(\sigma_{kg}^2 - m_{kg})} s_n \frac{(c_n b_g - 1)}{(\sigma_{kg}^2 - m_{kg})} \\
&= r_{kg} s_n \frac{(c_n b_g - 1)}{(\sigma_{kg}^2 - m_{kg})}
\end{aligned}
\tag{B.56}
$$

$$
\begin{aligned}
\frac{\partial}{\partial \mu_{kg}}(1-p_{kg}) &= \frac{\partial}{\partial \mu_{kg}}\frac{m_{kg}}{\sigma_{kg}^2} \\
&= \frac{m'_{kg}\sigma_{kg}^2 - m_{kg}(\sigma_{kg}^2)'}{\sigma_{kg}^4} \\
&= \frac{s_n\sigma_{kg}^2 - s_n\mu_{kg}(2c_n s_n^2 a_g \mu_{kg} + c_n s_n b_g)}{\sigma_{kg}^4} \\
&= \frac{s_n\sigma_{kg}^2 - s_n(2c_n s_n^2 a_g \mu_{kg}^2 + c_n s_n b_g \mu_{kg})}{\sigma_{kg}^4} \\
&= \frac{s_n\sigma_{kg}^2 - s_n(\sigma_{kg}^2 + c_n s_n^2 a_g \mu_{kg}^2)}{\sigma_{kg}^4} \\
&= \frac{s_n(\sigma_{kg}^2 - \sigma_{kg}^2 - c_n s_n^2 a_g \mu_{kg}^2)}{\sigma_{kg}^4} \\
&= -\frac{s_n^3 c_n a_g \mu_{kg}^2}{\sigma_{kg}^4} \\
&= -\frac{(s_n\mu_{kg})^2 c_n s_n a_g}{\sigma_{kg}^4} \\
&= -\frac{m_{kg}^2}{\sigma_{kg}^4} c_n s_n a_g \\
&= -(1-p_{kg})^2 c_n s_n a_g \qquad \text{(B.57)} \\
&= -\frac{\partial}{\partial \mu_{kg}} p_{kg} \qquad \text{(B.58)}
\end{aligned}
$$

and now continuing from B.37:

$$
\begin{aligned}
\frac{\partial}{\partial \mu_{kg}} \log \mathrm{NegBin}(x_{ng}|r_{ng}, p_{kg}) &= \frac{1}{\Gamma(x_{ng}+r_{kg})}\Gamma(x_{ng}+r_{kg})\psi^0(x_{ng}+r_{kg})\frac{\partial}{\partial \mu_{kg}} r_{kg} \\
&\quad -\frac{1}{\Gamma(r_{kg})}\Gamma(r_{kg})\psi^0(r_{kg})\frac{\partial}{\partial \mu_{kg}} r_{kg} + \log(1-p_{kg})\frac{\partial}{\partial \mu_{kg}} r_{kg} \\
&\quad + r_{kg}\frac{1}{(1-p_{kg})}\frac{\partial}{\partial \mu_{kg}}(1-p_{kg}) + x_{ng}\frac{1}{p_{kg}}\frac{\partial}{\partial \mu_{kg}} p_{kg} \\
&= \psi^0(x_{ng}+r_{kg})\frac{\partial r_{kg}}{\partial \mu_{kg}} - \psi^0(r_{kg})\frac{\partial r_{kg}}{\partial \mu_{kg}} + \log(1-p_{kg})\frac{\partial r_{kg}}{\partial \mu_{kg}} \\
&\quad - \frac{r_{kg}}{(1-p_{kg})}\frac{\partial}{\partial \mu_{kg}} p_{kg} + \frac{x_{ng}}{p_{kg}}\frac{\partial}{\partial \mu_{kg}} p_{kg} \\
&= \frac{\partial r_{kg}}{\partial \mu_{kg}}(\psi^0(x_{ng}+r_{kg}) - \psi^0(r_{kg}) + \log(1-p_{kg})) \\
&\quad + \left(\frac{x_{ng}}{p_{kg}} - \frac{r_{kg}}{1-p_{kg}}\right)\frac{\partial p_{kg}}{\partial \mu_{kg}} \qquad \text{(B.59)}
\end{aligned}
$$

C. Partial derivatives of bias and variance for optimal nearest neighbour smoothing

We need the partial derivatives of all main actors - $\widetilde{x}_{ig}, \widetilde{\sigma}^2_{ig}, \text{bias}(\vec{W}_i), \text{var}(\vec{W}_i)$. As in the main text, when writing $\sum_j$ we will mean $\sum_{j \in \text{NN}_i}$ and when writing $\sum_g$ we will mean $\sum_{g=1}^{G}$.

- The derivative of $\widetilde{x}_{ig}$, since it comes up in all formulas:

$$\begin{aligned}\frac{\partial \widetilde{x}_{ig}}{\partial w_j} &= \frac{x_{jg}}{\sum_j w_j} - \frac{\sum_j w_j x_{jg}}{(\sum_j w_j)^2} \\ &= \frac{x_{jg} - \widetilde{x}_{ig}}{\sum_j w_j}\end{aligned}$$

$$\boxed{\frac{\partial \widetilde{x}_{ig}}{\partial w_j} = \frac{x_{jg} - \widetilde{x}_{ig}}{\sum_j w_j}} \tag{C.1}$$

- The second helper derivative is $\widetilde{\sigma}^2_{ig}$.

$$\boxed{\frac{\partial \widetilde{\sigma}^2_{ig}}{\partial w_j} = \frac{(x_{ig} - \widetilde{x}_{ig})^2 - \widetilde{\sigma}^2_{ig}}{\sum_j w_j}} \tag{C.2}$$

- The derivative of the bias:

$$\boxed{\frac{\partial \text{bias}}{\partial w_j} = \frac{1}{2\sum_j w_j} \sum_g -\frac{2(x_{ig} - \widetilde{x}_{ig})(x_{jg} - \widetilde{x}_{ig})}{\widetilde{\sigma}^2_{ig}} - \frac{(x_{ig} - \widetilde{x}_{ig})^2 (x_{jg} - \widetilde{x}_{ig})^2}{\widetilde{\sigma}^4_{ig}} + \frac{(x_{ig} - \widetilde{x}_{ig})^2}{\widetilde{\sigma}^2_{ig}}} \tag{C.3}$$

- and the derivative of the variance term:

$$\boxed{\frac{\partial \widetilde{\sigma}^2_{ig}}{\partial w_j} = \frac{G w_j}{(\sum_j w_j)^2} - \frac{G(\sum_j {w_j}^2)}{(\sum_j w_j)^3}} \tag{C.4}$$

References

[1] Adam, M., Potter, A. S., and Potter, S. S. (2017). Psychrophilic proteases dramatically reduce single-cell rna-seq artifacts: a molecular atlas of kidney development. *Development*, **144**(19), 3625–3632.

[2] Adamson, B., Norman, T. M., Jost, M., Cho, M. Y., Nuñez, J. K., Chen, Y., Villalta, J. E., Gilbert, L. A., Horlbeck, M. A., Hein, M. Y., *et al.* (2016). A multiplexed single-cell crispr screening platform enables systematic dissection of the unfolded protein response. *Cell*, **167**(7), 1867–1882.

[3] Ahmed, S., Rattray, M., and Boukouvalas, A. (2018). Grandprix: Scaling up the bayesian gplvm for single-cell data. *Bioinformatics*, **35**(1), 47–54.

[4] Alemany, A., Florescu, M., Baron, C. S., Peterson-Maduro, J., and Van Oudenaarden, A. (2018). Whole-organism clone tracing using single-cell sequencing. *Nature*, **556**(7699), 108.

[5] Alles, J., Karaiskos, N., Praktiknjo, S. D., Grosswendt, S., Wahle, P., Ruffault, P.-L., Ayoub, S., Schreyer, L., Boltengagen, A., Birchmeier, C., *et al.* (2017). Cell fixation and preservation for droplet-based single-cell transcriptomics. *BMC biology*, **15**(1), 44.

[6] Amodio, M., Van Dijk, D., Srinivasan, K., Chen, W. S., Mohsen, H., Moon, K. R., Campbell, A., Zhao, Y., Wang, X., Venkataswamy, M., *et al.* (2019). Exploring single-cell data with deep multitasking neural networks. *BioRxiv*, page 237065.

[7] Amrhein, L., Harsha, K., and Fuchs, C. (2019). A mechanistic model for the negative binomial distribution of single-cell mrna counts. *bioRxiv*, page 657619.

[8] Andersen, A.-M. N., Wohlfahrt, J., Christens, P., Olsen, J., and Melbye, M. (2000). Maternal age and fetal loss: population based register linkage study. *Bmj*, **320**(7251), 1708–1712.

[9] Andor, N., Simonds, E. F., Czerwinski, D. K., Chen, J., Grimes, S. M., Wood-Bouwens, C., Zheng, G. X. Y., Kubit, M. A., Greer, S., Weiss, W. A., Levy, R., and Ji, H. P. (2018). Single-cell rna-seq of lymphoma cancers reveals malignant b cell types and co-expression of t cell immune checkpoints. *Blood*.

[10] Angerer, P., Haghverdi, L., Büttner, M., Theis, F. J., Marr, C., and Buettner, F. (2016). destiny: diffusion maps for large-scale single-cell data in r. *Bioinformatics*, **32**(8), 1241–1243.

[11] Athanasiadis, E. I., Botthof, J. G., Andres, H., Ferreira, L., Lio, P., and Cvejic, A. (2017). Single-cell rna-sequencing uncovers transcriptional states and fate decisions in haematopoiesis. *Nature communications*, **8**(1), 2045.

[12] Bai, Y.-L., Baddoo, M., Flemington, E. K., Nakhoul, H. N., and Liu, Y.-Z. (2019). Screen technical noise in single cell rna sequencing data. *Genomics*.

[13] Becht, E., McInnes, L., Healy, J., Dutertre, C.-A., Kwok, I. W., Ng, L. G., Ginhoux, F., and Newell, E. W. (2019). Dimensionality reduction for visualizing single-cell data using umap. *Nature biotechnology*, **37**(1), 38.

[14] Bendall, S. C., Davis, K. L., Amir, E.-a. D., Tadmor, M. D., Simonds, E. F., Chen, T. J., Shenfeld, D. K., Nolan, G. P., and Pe'er, D. (2014). Single-cell trajectory detection uncovers progression and regulatory coordination in human b cell development. *Cell*, **157**(3), 714–725.

[15] Bose, S., Wan, Z., Carr, A., Rizvi, A. H., Vieira, G., Pe'er, D., and Sims, P. A. (2015). Scalable microfluidics for single-cell rna printing and sequencing. *Genome biology*, **16**(1), 120.

[16] Braga, F. A. V., Kar, G., Berg, M., Carpaij, O. A., Polanski, K., Simon, L. M., Brouwer, S., Gomes, T., Hesse, L., Jiang, J., *et al.* (2019). A cellular census of healthy lung and asthmatic airway wall identifies novel cell states in health and disease. *bioRxiv*, page 527408.

[17] Brennecke, P., Anders, S., Kim, J. K., Kołodziejczyk, A. A., Zhang, X., Proserpio, V., Baying, B., Benes, V., Teichmann, S. A., Marioni, J. C., *et al.* (2013). Accounting for technical noise in single-cell rna-seq experiments. *Nature methods*, **10**(11), 1093.

[18] Briggs, J. A., Weinreb, C., Wagner, D. E., Megason, S., Peshkin, L., Kirschner, M. W., and Klein, A. M. (2018). The dynamics of gene expression in vertebrate embryogenesis at single-cell resolution. *Science*, **360**(6392), eaar5780.

[19] Brouilette, S., Kuersten, S., Mein, C., Bozek, M., Terry, A., Dias, K.-R., Bhaw-Rosun, L., Shintani, Y., Coppen, S., Ikebe, C., *et al.* (2012). A simple and novel method for rna-seq library preparation of single cell cdna analysis by hyperactive tn5 transposase. *Developmental Dynamics*, **241**(10), 1584–1590.

[20] Buettner, F., Natarajan, K. N., Casale, F. P., Proserpio, V., Scialdone, A., Theis, F. J., Teichmann, S. A., Marioni, J. C., and Stegle, O. (2015). Computational analysis of cell-to-cell heterogeneity in single-cell rna-sequencing data reveals hidden subpopulations of cells. *Nature biotechnology*, **33**(2), 155.

[21] Burdziak, C., Azizi, E., Prabhakaran, S., and Pe'er, D. (2019). A nonparametric multi-view model for estimating cell type-specific gene regulatory networks. *arXiv preprint arXiv:1902.08138*.

[22] Butler, A., Hoffman, P., Smibert, P., Papalexi, E., and Satija, R. (2018). Integrating single-cell transcriptomic data across different conditions, technologies, and species. *Nature biotechnology*, **36**(5), 411.

[23] Cannoodt, R., Saelens, W., and Saeys, Y. (2016). Computational methods for trajectory inference from single-cell transcriptomics. *European journal of immunology*, **46**(11), 2496–2506.

[24] Cao, J., Packer, J. S., Ramani, V., Cusanovich, D. A., Huynh, C., Daza, R., Qiu, X., Lee, C., Furlan, S. N., Steemers, F. J., *et al.* (2017). Comprehensive single-cell transcriptional profiling of a multicellular organism. *Science*, **357**(6352), 661–667.

[25] Chen, R., Wu, X., Jiang, L., and Zhang, Y. (2017). Single-cell rna-seq reveals hypothalamic cell diversity. *Cell reports*, **18**(13), 3227–3241.

[26] Ching, T., Himmelstein, D. S., Beaulieu-Jones, B. K., Kalinin, A. A., Do, B. T., Way, G. P., Ferrero, E., Agapow, P.-M., Zietz, M., Hoffman, M. M., *et al.* (2018). Opportunities and obstacles for deep learning in biology and medicine. *Journal of The Royal Society Interface*, **15**(141), 20170387.

[27] Coifman, R. R., Lafon, S., Lee, A. B., Maggioni, M., Nadler, B., Warner, F., and Zucker, S. W. (2005). Geometric diffusions as a tool for harmonic analysis and structure definition of data: Diffusion maps. *Proceedings of the National Academy of Sciences of the United States of America*, **102**(21), 7426–7431.

[28] Collin, J., Queen, R., Zerti, D., Dorgau, B., Hussain, R., Coxhead, J., Cockell, S., and Lako, M. (2019). Deconstructing retinal organoids: Single cell rna-seq reveals the cellular components of human pluripotent stem cell-derived retina. *Stem Cells*, **37**(5), 593–598.

[29] Darmanis, S., Sloan, S. A., Zhang, Y., Enge, M., Caneda, C., Shuer, L. M., Gephart, M. G. H., Barres, B. A., and Quake, S. R. (2015). A survey of human brain transcriptome diversity at the single cell level. *Proceedings of the National Academy of Sciences*, **112**(23), 7285–7290.

[30] Darmanis, S., Sloan, S. A., Croote, D., Mignardi, M., Chernikova, S., Samghababi, P., Zhang, Y., Neff, N., Kowarsky, M., Caneda, C., *et al.* (2017). Single-cell rna-seq analysis of infiltrating neoplastic cells at the migrating front of human glioblastoma. *Cell reports*, **21**(5), 1399–1410.

[31] Datlinger, P., Rendeiro, A. F., Schmidl, C., Krausgruber, T., Traxler, P., Klughammer, J., Schuster, L. C., Kuchler, A., Alpar, D., and Bock, C. (2017). Pooled crispr screening with single-cell transcriptome readout. *Nature methods*, **14**(3), 297.

[32] Ding, J., Condon, A., and Shah, S. P. (2018). Interpretable dimensionality reduction of single cell transcriptome data with deep generative models. *Nature communications*, **9**(1), 2002.

[33] Dixit, A., Parnas, O., Li, B., Chen, J., Fulco, C. P., Jerby-Arnon, L., Marjanovic, N. D., Dionne, D., Burks, T., Raychowdhury, R., *et al.* (2016). Perturb-seq: dissecting molecular circuits with scalable single-cell rna profiling of pooled genetic screens. *Cell*, **167**(7), 1853–1866.

[34] Duane, S., Kennedy, A. D., Pendleton, B. J., and Roweth, D. (1987). Hybrid monte carlo. *Physics letters B*, **195**(2), 216–222.

[35] Ellwanger, D. C., Scheibinger, M., Dumont, R. A., Barr-Gillespie, P. G., and Heller, S. (2018). Transcriptional dynamics of hair-bundle morphogenesis revealed with celltrails. *Cell reports*, **23**(10), 2901–2914.

[36] Elowitz, M. B., Levine, A. J., Siggia, E. D., and Swain, P. S. (2002). Stochastic gene expression in a single cell. *Science*, **297**(5584), 1183–1186.

[37] Elyanow, R., Dumitrascu, B., Engelhardt, B. E., and Raphael, B. J. (2019). netnmf-sc: A network regularization algorithm for dimensionality reduction and imputation of single-cell expression data. In *RECOMB*, pages 297–298. Springer.

[38] Eraslan, G., Simon, L. M., Mircea, M., Mueller, N. S., and Theis, F. J. (2019). Single-cell rna-seq denoising using a deep count autoencoder. *Nature communications*, **10**(1), 390.

[39] Fan, H. C., Fu, G. K., and Fodor, S. P. (2015). Combinatorial labeling of single cells for gene expression cytometry. *Science*, **347**(6222), 1258367.

[40] Fan, X., Dong, J., Zhong, S., Wei, Y., Wu, Q., Yan, L., Yong, J., Sun, L., Wang, X., Zhao, Y., *et al.* (2018). Spatial transcriptomic survey of human embryonic cerebral cortex by single-cell rna-seq analysis. *Cell research*, **28**(7), 730.

[41] Farrell, J. A., Wang, Y., Riesenfeld, S. J., Shekhar, K., Regev, A., and Schier, A. F. (2018a). Single-cell reconstruction of developmental trajectories during zebrafish embryogenesis. *Science*, **360**(6392), eaar3131.

[42] Farrell, J. A., Wang, Y., Riesenfeld, S. J., Shekhar, K., Regev, A., and Schier, A. F. (2018b). Single-cell reconstruction of developmental trajectories during zebrafish embryogenesis. *Science*, **360**(6392), eaar3131.

[43] Filbin, M. G., Tirosh, I., Hovestadt, V., Shaw, M. L., Escalante, L. E., Mathewson, N. D., Neftel, C., Frank, N., Pelton, K., Hebert, C. M., *et al.* (2018). Developmental and oncogenic programs in h3k27m gliomas dissected by single-cell rna-seq. *Science*, **360**(6386), 331–335.

[44] Francesconi, M., Di Stefano, B., Berenguer, C., de Andrés-Aguayo, L., Plana-Carmona, M., Mendez-Lago, M., Guillaumet-Adkins, A., Rodriguez-Esteban, G., Gut, M., Gut, I. G., *et al.* (2019). Single cell rna-seq identifies the origins of heterogeneity in efficient cell transdifferentiation and reprogramming. *eLife*, **8**, e41627.

[45] Gerber, T., Murawala, P., Knapp, D., Masselink, W., Schuez, M., Hermann, S., Gac-Santel, M., Nowoshilow, S., Kageyama, J., Khattak, S., *et al.* (2018). Single-cell analysis uncovers convergence of cell identities during axolotl limb regeneration. *Science*, page eaaq0681.

[46] Gierahn, T. M., Wadsworth II, M. H., Hughes, T. K., Bryson, B. D., Butler, A., Satija, R., Fortune, S., Love, J. C., and Shalek, A. K. (2017). Seq-well: portable, low-cost rna sequencing of single cells at high throughput. *Nature methods*, **14**(4), 395.

[47] Gilbert, S. F. (2000). *Developmental Biology*. Sinauer Associates, Sunderland (MA), 6th edition edition. Available from: https://www.ncbi.nlm.nih.gov/books/NBK9983/.

[48] Gorban, A. and Zinovyev, A. (2005). Elastic principal graphs and manifolds and their practical applications. *Computing*, **75**(4), 359–379.

[49] Green, C. D., Ma, Q., Manske, G. L., Shami, A. N., Zheng, X., Marini, S., Moritz, L., Sultan, C., Gurczynski, S. J., Moore, B. B., *et al.* (2018). A comprehensive roadmap of murine spermatogenesis defined by single-cell rna-seq. *Developmental cell*, **46**(5), 651–667.

[50] Grindberg, R. V., Yee-Greenbaum, J. L., McConnell, M. J., Novotny, M., O'Shaughnessy, A. L., Lambert, G. M., Araúzo-Bravo, M. J., Lee, J., Fishman, M., Robbins, G. E., *et al.* (2013). Rna-sequencing from single nuclei. *Proceedings of the National Academy of Sciences*, **110**(49), 19802–19807.

[51] Grün, D., Kester, L., and Van Oudenaarden, A. (2014). Validation of noise models for single-cell transcriptomics. *Nature methods*, **11**(6), 637.

[52] Grün, D., Lyubimova, A., Kester, L., Wiebrands, K., Basak, O., Sasaki, N., Clevers, H., and van Oudenaarden, A. (2015). Single-cell messenger rna sequencing reveals rare intestinal cell types. *Nature*, **525**(7568), 251.

[53] Grün, D., Muraro, M. J., Boisset, J.-C., Wiebrands, K., Lyubimova, A., Dharmadhikari, G., van den Born, M., van Es, J., Jansen, E., Clevers, H., *et al.* (2016). De novo prediction of stem cell identity using single-cell transcriptome data. *Cell Stem Cell*, **19**(2), 266–277.

[54] Guillaumet-Adkins, A., Rodríguez-Esteban, G., Mereu, E., Mendez-Lago, M., Jaitin, D. A., Villanueva, A., Vidal, A., Martinez-Marti, A., Felip, E., Vivancos, A., *et al.* (2017). Single-cell transcriptome conservation in cryopreserved cells and tissues. *Genome biology*, **18**(1), 45.

[55] Guo, J., Grow, E. J., Yi, C., Mlcochova, H., Maher, G. J., Lindskog, C., Murphy, P. J., Wike, C. L., Carrell, D. T., Goriely, A., *et al.* (2017). Chromatin and single-cell rna-seq profiling reveal dynamic signaling and metabolic transitions during human spermatogonial stem cell development. *Cell Stem Cell*, **21**(4), 533–546.

[56] Haber, A. L., Biton, M., Rogel, N., Herbst, R. H., Shekhar, K., Smillie, C., Burgin, G., Delorey, T. M., Howitt, M. R., Katz, Y., *et al.* (2017). A single-cell survey of the small intestinal epithelium. *Nature*, **551**(7680), 333.

[57] Habib, N., Li, Y., Heidenreich, M., Swiech, L., Avraham-Davidi, I., Trombetta, J. J., Hession, C., Zhang, F., and Regev, A. (2016). Div-seq: Single-nucleus rna-seq reveals dynamics of rare adult newborn neurons. *Science*, **353**(6302), 925–928.

[58] Habib, N., Avraham-Davidi, I., Basu, A., Burks, T., Shekhar, K., Hofree, M., Choudhury, S. R., Aguet, F., Gelfand, E., Ardlie, K., *et al.* (2017). Massively parallel single-nucleus rna-seq with dronc-seq. *Nature methods*, **14**(10), 955.

[59] Haghverdi, L., Buettner, F., and Theis, F. J. (2015). Diffusion maps for high-dimensional single-cell analysis of differentiation data. *Bioinformatics*, **31**(18), 2989–2998.

[60] Haghverdi, L., Lun, A. T., Morgan, M. D., and Marioni, J. C. (2018). Batch effects in single-cell rna-sequencing data are corrected by matching mutual nearest neighbors. *Nature biotechnology*, **36**(5), 421.

[61] Halpern, K. B., Shenhav, R., Matcovitch-Natan, O., Tóth, B., Lemze, D., Golan, M., Massasa, E. E., Baydatch, S., Landen, S., Moor, A. E., *et al.* (2017). Single-cell spatial reconstruction reveals global division of labour in the mammalian liver. *Nature*, **542**(7641), 352.

[62] Hammond, T. R., Dufort, C., Dissing-Olesen, L., Giera, S., Young, A., Wysoker, A., Walker, A. J., Gergits, F., Segel, M., Nemesh, J., *et al.* (2019). Single-cell rna sequencing of microglia throughout the mouse lifespan and in the injured brain reveals complex cell-state changes. *Immunity*, **50**(1), 253–271.

[63] Han, X., Wang, R., Zhou, Y., Fei, L., Sun, H., Lai, S., Saadatpour, A., Zhou, Z., Chen, H., Ye, F., *et al.* (2018). Mapping the mouse cell atlas by microwell-seq. *Cell*, **172**(5), 1091–1107.

[64] Hashimshony, T., Wagner, F., Sher, N., and Yanai, I. (2012). Cel-seq: single-cell rna-seq by multiplexed linear amplification. *Cell reports*, **2**(3), 666–673.

[65] Hashimshony, T., Senderovich, N., Avital, G., Klochendler, A., de Leeuw, Y., Anavy, L., Gennert, D., Li, S., Livak, K. J., Rozenblatt-Rosen, O., *et al.* (2016). Cel-seq2: sensitive highly-multiplexed single-cell rna-seq. *Genome biology*, **17**(1), 77.

[66] Herman, J. S., Grün, D., *et al.* (2018). Fateid infers cell fate bias in multipotent progenitors from single-cell rna-seq data. *Nature methods*, **15**(5), 379.

[67] Hochgerner, H., Zeisel, A., Lönnerberg, P., and Linnarsson, S. (2018). Conserved properties of dentate gyrus neurogenesis across postnatal development revealed by single-cell rna sequencing. *Nature neuroscience*, **21**(2), 290.

[68] Hu, P., Fabyanic, E., Kwon, D. Y., Tang, S., Zhou, Z., and Wu, H. (2017). Dissecting cell-type composition and activity-dependent transcriptional state in mammalian brains by massively parallel single-nucleus rna-seq. *Molecular cell*, **68**(5), 1006–1015.

[69] Huang, M., Wang, J., Torre, E., Dueck, H., Shaffer, S., Bonasio, R., Murray, J. I., Raj, A., Li, M., and Zhang, N. R. (2018). Saver: gene expression recovery for single-cell rna sequencing. *Nature methods*, **15**(7), 539.

[70] Hyvärinen, A. and Oja, E. (2000). Independent component analysis: algorithms and applications. *Neural networks*, **13**(4-5), 411–430.

[71] Islam, S., Kjällquist, U., Moliner, A., Zajac, P., Fan, J.-B., Lönnerberg, P., and Linnarsson, S. (2011). Characterization of the single-cell transcriptional landscape by highly multiplex rna-seq. *Genome research*, **21**(7), 1160–1167.

[72] Islam, S., Zeisel, A., Joost, S., La Manno, G., Zajac, P., Kasper, M., Lönnerberg, P., and Linnarsson, S. (2014). Quantitative single-cell rna-seq with unique molecular identifiers. *Nature methods*, **11**(2), 163.

[73] Jaitin, D. A., Kenigsberg, E., Keren-Shaul, H., Elefant, N., Paul, F., Zaretsky, I., Mildner, A., Cohen, N., Jung, S., Tanay, A., *et al.* (2014). Massively parallel single-cell rna-seq for marker-free decomposition of tissues into cell types. *Science*, **343**(6172), 776–779.

[74] Jaitin, D. A., Weiner, A., Yofe, I., Lara-Astiaso, D., Keren-Shaul, H., David, E., Salame, T. M., Tanay, A., van Oudenaarden, A., and Amit, I. (2016). Dissecting immune circuits by linking crispr-pooled screens with single-cell rna-seq. *Cell*, **167**(7), 1883–1896.

[75] Ji, Q., Zheng, Y., Zhang, G., Hu, Y., Fan, X., Hou, Y., Wen, L., Li, L., Xu, Y., Wang, Y., *et al.* (2019). Single-cell rna-seq analysis reveals the progression of human osteoarthritis. *Annals of the rheumatic diseases*, **78**(1), 100–110.

[76] Ji, Z. and Ji, H. (2016). Tscan: Pseudo-time reconstruction and evaluation in single-cell rna-seq analysis. *Nucleic acids research*, **44**(13), e117–e117.

[77] Jiang, L., Schlesinger, F., Davis, C. A., Zhang, Y., Li, R., Salit, M., Gingeras, T. R., and Oliver, B. (2011). Synthetic spike-in standards for rna-seq experiments. *Genome research*, **21**(9), 1543–1551.

[78] Johnson, J., Canning, J., Kaneko, T., Pru, J. K., and Tilly, J. L. (2004). Germline stem cells and follicular renewal in the postnatal mammalian ovary. *Nature*, **428**(6979), 145.

[79] Johnson, J., Bagley, J., Skaznik-Wikiel, M., Lee, H.-J., Adams, G. B., Niikura, Y., Tschudy, K. S., Tilly, J. C., Cortes, M. L., Forkert, R., *et al.* (2005). Oocyte generation in adult mammalian ovaries by putative germ cells in bone marrow and peripheral blood. *Cell*, **122**(2), 303–315.

[80] Jolliffe, I. (2011). *Principal component analysis*. Springer.

[81] Joost, S., Zeisel, A., Jacob, T., Sun, X., La Manno, G., Lönnerberg, P., Linnarsson, S., and Kasper, M. (2016). Single-cell transcriptomics reveals that differentiation and spatial signatures shape epidermal and hair follicle heterogeneity. *Cell systems*, **3**(3), 221–237.

[82] Jordão, M. J. C., Sankowski, R., Brendecke, S. M., Locatelli, G., Tai, Y.-H., Tay, T. L., Schramm, E., Armbruster, S., Hagemeyer, N., Groß, O., *et al.* (2019). Single-cell profiling identifies myeloid cell subsets with distinct fates during neuroinflammation. *Science*, **363**(6425), eaat7554.

[83] Junker, J. P., Spanjaard, B., Peterson-Maduro, J., Alemany, A., Hu, B., Florescu, M., and van Oudenaarden, A. (2016). Massively parallel whole-organism lineage tracing using crispr/cas9 induced genetic scars. *BioRxiv*, page 056499.

[84] Kang, H. M., Subramaniam, M., Targ, S., Nguyen, M., Maliskova, L., McCarthy, E., Wan, E., Wong, S., Byrnes, L., Lanata, C. M., *et al.* (2018). Multiplexed droplet single-cell rna-sequencing using natural genetic variation. *Nature biotechnology*, **36**(1), 89.

[85] Karaayvaz, M., Cristea, S., Gillespie, S. M., Patel, A. P., Mylvaganam, R., Luo, C. C., Specht, M. C., Bernstein, B. E., Michor, F., and Ellisen, L. W. (2018). Unravelling subclonal heterogeneity and aggressive disease states in tnbc through single-cell rna-seq. *Nature communications*, **9**(1), 3588.

[86] Karaiskos, N., Wahle, P., Alles, J., Boltengagen, A., Ayoub, S., Kipar, C., Kocks, C., Rajewsky, N., and Zinzen, R. P. (2017). The drosophila embryo at single-cell transcriptome resolution. *Science*, **358**(6360), 194–199.

[87] Kim, C., Gao, R., Sei, E., Brandt, R., Hartman, J., Hatschek, T., Crosetto, N., Foukakis, T., and Navin, N. E. (2018). Chemoresistance evolution in triple-negative breast cancer delineated by single-cell sequencing. *Cell*, **173**(4), 879–893.

[88] Kim, J. K. and Marioni, J. C. (2013). Inferring the kinetics of stochastic gene expression from single-cell rna-sequencing data. *Genome biology*, **14**(1), R7.

[89] Klein, A. M., Mazutis, L., Akartuna, I., Tallapragada, N., Veres, A., Li, V., Peshkin, L., Weitz, D. A., and Kirschner, M. W. (2015). Droplet barcoding for single-cell transcriptomics applied to embryonic stem cells. *Cell*, **161**(5), 1187–1201.

[90] Kolodziejczyk, A., Kim, J., Tsang, J., Ilicic, T., Henriksson, J., Natarajan, K., Tuck, A., Gao, X., BÃ¼hler, M., Liu, P., and Marioni, J. (2015). Scalable microfluidics for single-cell rna printing and sequencing. *Cell stem cell*, **17**(4), 471–485.

[91] Korsunsky, I., Fan, J., Slowikowski, K., Zhang, F., Wei, K., Baglaenko, Y., Brenner, M., Loh, P.-R., and Raychaudhuri, S. (2018). Fast, sensitive, and flexible integration of single cell data with harmony. *BioRxiv*, page 461954.

[92] Kumar, R. M., Cahan, P., Shalek, A. K., Satija, R., DaleyKeyser, A. J., Li, H., Zhang, J., Pardee, K., Gennert, D., Trombetta, J. J., *et al.* (2014). Deconstructing transcriptional heterogeneity in pluripotent stem cells. *Nature*, **516**(7529), 56.

[93] La Manno, G., Gyllborg, D., Codeluppi, S., Nishimura, K., Salto, C., Zeisel, A., Borm, L. E., Stott, S. R., Toledo, E. M., Villaescusa, J. C., *et al.* (2016). Molecular diversity of midbrain development in mouse, human, and stem cells. *Cell*, **167**(2), 566–580.

[94] La Manno, G., Soldatov, R., Zeisel, A., Braun, E., Hochgerner, H., Petukhov, V., Lidschreiber, K., Kastriti, M. E., Lönnerberg, P., Furlan, A., *et al.* (2018). Rna velocity of single cells. *Nature*, **560**(7719), 494.

[95] Lake, B. B., Ai, R., Kaeser, G. E., Salathia, N. S., Yung, Y. C., Liu, R., Wildberg, A., Gao, D., Fung, H.-L., Chen, S., *et al.* (2016). Neuronal subtypes and diversity revealed by single-nucleus rna sequencing of the human brain. *Science*, **352**(6293), 1586–1590.

[96] Li, L., Dong, J., Yan, L., Yong, J., Liu, X., Hu, Y., Fan, X., Wu, X., Guo, H., Wang, X., *et al.* (2017). Single-cell rna-seq analysis maps development of human germline cells and gonadal niche interactions. *Cell Stem Cell*, **20**(6), 858–873.

[97] Li, W. V. and Li, J. J. (2018). An accurate and robust imputation method scimpute for single-cell rna-seq data. *Nature communications*, **9**(1), 997.

[98] Li, X., Zhang, S., and Wong, K.-C. (2018). Single-cell rna-seq interpretations using evolutionary multiobjective ensemble pruning. *Bioinformatics*.

[99] Lin, P., Troup, M., and Ho, J. W. (2017). Cidr: Ultrafast and accurate clustering through imputation for single-cell rna-seq data. *Genome biology*, **18**(1), 59.

[100] Linderman, G. C., Zhao, J., and Kluger, Y. (2018). Zero-preserving imputation of scrna-seq data using low-rank approximation. *bioRxiv*, page 397588.

[101] Lun, A. T. and Marioni, J. C. (2017). Overcoming confounding plate effects in differential expression analyses of single-cell rna-seq data. *Biostatistics*, **18**(3), 451–464.

[102] Lun, A. T., Bach, K., and Marioni, J. C. (2016). Pooling across cells to normalize single-cell rna sequencing data with many zero counts. *Genome biology*, **17**(1), 75.

[103] Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of Machine Learning Research*, **9**(Nov), 2579–2605.

[104] Macosko, E. Z., Basu, A., Satija, R., Nemesh, J., Shekhar, K., Goldman, M., Tirosh, I., Bialas, A. R., Kamitaki, N., Martersteck, E. M., *et al.* (2015). Highly parallel genome-wide expression profiling of individual cells using nanoliter droplets. *Cell*, **161**(5), 1202–1214.

[105] Magnúsdóttir, E., Dietmann, S., Murakami, K., Günesdogan, U., Tang, F., Bao, S., Diamanti, E., Lao, K., Gottgens, B., and Surani, M. A. (2013). A tripartite transcription factor network regulates primordial germ cell specification in mice. *Nature cell biology*, **15**(8), 905.

[106] Mao, Q., Wang, L., Goodison, S., and Sun, Y. (2015). Dimensionality reduction via graph structure learning. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 765–774, New York, NY, USA. ACM.

[107] Mao, Q., Wang, L., Tsang, I. W., and Sun, Y. (2016). Principal graph and structure learning based on reversed graph embedding. *IEEE transactions on pattern analysis and machine intelligence*, **39**(11), 2227–2241.

[108] Marinov, G. K., Williams, B. A., McCue, K., Schroth, G. P., Gertz, J., Myers, R. M., and Wold, B. J. (2014). From single-cell to cell-pool transcriptomes: stochasticity in gene expression and rna splicing. *Genome research*, **24**(3), 496–510.

[109] Marques, S., Zeisel, A., Codeluppi, S., van Bruggen, D., Falcão, A. M., Xiao, L., Li, H., Häring, M., Hochgerner, H., Romanov, R. A., *et al.* (2016). Oligodendrocyte heterogeneity in the mouse juvenile and adult central nervous system. *Science*, **352**(6291), 1326–1329.

[110] Matsumoto, H., Hayashi, T., Ozaki, H., Tsuyuzaki, K., Umeda, M., Iida, T., Nakamura, M., Okano, H., and Nikaido, I. (2019). A nmf-based approach to discover overlooked differentially expressed gene regions from single-cell rna-seq data. *BioRxiv*, page 543447.

[111] McCarthy, D. J., Campbell, K. R., Lun, A. T., and Wills, Q. F. (2017). Scater: pre-processing, quality control, normalization and visualization of single-cell rna-seq data in r. *Bioinformatics*, **33**(8), 1179–1186.

[112] McInnes, L., Healy, J., and Melville, J. (2018). Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*.

[113] McKenna, A., Findlay, G. M., Gagnon, J. A., Horwitz, M. S., Schier, A. F., and Shendure, J. (2016). Whole-organism lineage tracing by combinatorial and cumulative genome editing. *Science*, **353**(6298), aaf7907.

[114] Moignard, V., Woodhouse, S., Haghverdi, L., Lilly, A. J., Tanaka, Y., Wilkinson, A. C., Buettner, F., Macaulay, I. C., Jawaid, W., Diamanti, E., *et al.* (2015). Decoding the regulatory network of early blood development from single-cell gene expression measurements. *Nature biotechnology*, **33**(3), 269.

[115] Moon, K. R., van Dijk, D., Wang, Z., Chen, W., Hirn, M. J., Coifman, R. R., Ivanova, N. B., Wolf, G., and Krishnaswamy, S. (2017). Phate: a dimensionality reduction method for visualizing trajectory structures in high-dimensional biological data. *bioRxiv*, page 120378.

[116] Muraro, M. J., Dharmadhikari, G., Grün, D., Groen, N., Dielen, T., Jansen, E., van Gurp, L., Engelse, M. A., Carlotti, F., de Koning, E. J., *et al.* (2016). A single-cell transcriptome atlas of the human pancreas. *Cell systems*, **3**(4), 385–394.

[117] Nelson, L. M. (2009). Primary ovarian insufficiency. *New England Journal of Medicine*, **360**(6), 606–614.

[118] Nguyen, Q. H., Pervolarakis, N., Blake, K., Ma, D., Davis, R. T., James, N., Phung, A. T., Willey, E., Kumar, R., Jabart, E., *et al.* (2018). Profiling human breast epithelial cells using single cell rna sequencing identifies cell diversity. *Nature communications*, **9**(1), 2028.

[119] O'Leary, N. A., Wright, M. W., Brister, J. R., Ciufo, S., Haddad, D., McVeigh, R., Rajput, B., Robbertse, B., Smith-White, B., Ako-Adjei, D., *et al.* (2015). Reference sequence (refseq) database at ncbi: current status, taxonomic expansion, and functional annotation. *Nucleic acids research*, **44**(D1), D733–D745.

[120] Pandey, S., Shekhar, K., Regev, A., and Schier, A. F. (2018). Comprehensive identification and spatial mapping of habenular neuronal types using single-cell rna-seq. *Current Biology*, **28**(7), 1052–1065.

[121] Papadopoulos, N., Gonzalo, P. R., and SÃ¶ding, J. (2019). PROSSTT: probabilistic simulation of single-cell RNA-seq data for complex differentiation processes. *Bioinformatics*.

[122] Park, J., Shrestha, R., Qiu, C., Kondo, A., Huang, S., Werth, M., Li, M., Barasch, J., and Suszták, K. (2018). Single-cell transcriptomics of the mouse kidney reveals potential cellular targets of kidney disease. *Science*, **360**(6390), 758–763.

[123] Parra, R. G., Papadopoulos, N., Ahumada-Arranz, L., El Kholtei, J., Treutlein, B., and Soeding, J. (2018). Reconstructing complex lineage trees from scRNA-seq data using MERLoT. *bioRxiv*.

[124] Patel, A. P., Tirosh, I., Trombetta, J. J., Shalek, A. K., Gillespie, S. M., Wakimoto, H., Cahill, D. P., Nahed, B. V., Curry, W. T., Martuza, R. L., *et al.* (2014). Single-cell rna-seq highlights intratumoral heterogeneity in primary glioblastoma. *Science*, **344**(6190), 1396–1401.

[125] Paul, F., Arkin, Y., Giladi, A., Jaitin, D. A., Kenigsberg, E., Keren-Shaul, H., Winter, D., Lara-Astiaso, D., Gury, M., Weiner, A., *et al.* (2015). Transcriptional heterogeneity and lineage commitment in myeloid progenitors. *Cell*, **163**(7), 1663–1677.

[126] Picelli, S., Björklund, Å. K., Faridani, O. R., Sagasser, S., Winberg, G., and Sandberg, R. (2013). Smart-seq2 for sensitive full-length transcriptome profiling in single cells. *Nature methods*, **10**(11), 1096.

[127] Pierson, E. and Yau, C. (2015). Zifa: Dimensionality reduction for zero-inflated single-cell gene expression analysis. *Genome biology*, **16**(1), 241.

[128] Plass, M., Solana, J., Wolf, F. A., Ayoub, S., Misios, A., Glažar, P., Obermayer, B., Theis, F. J., Kocks, C., and Rajewsky, N. (2018). Cell type atlas and lineage tree of a whole complex animal by single-cell transcriptomics. *Science*, **360**(6391), eaaq1723.

[129] Poran, A., Nötzel, C., Aly, O., Mencia-Trinchant, N., Harris, C. T., Guzman, M. L., Hassane, D. C., Elemento, O., and Kafsack, B. F. (2017). Single-cell rna sequencing reveals a signature of sexual commitment in malaria parasites. *Nature*, **551**(7678), 95.

[130] Qiu, X., Mao, Q., Tang, Y., Wang, L., Chawla, R., Pliner, H. A., and Trapnell, C. (2017). Reversed graph embedding resolves complex single-cell trajectories. *Nature methods*, **14**(10), 979–982.

[131] Raj, B., Wagner, D. E., McKenna, A., Pandey, S., Klein, A. M., Shendure, J., Gagnon, J. A., and Schier, A. F. (2018). Simultaneous single-cell profiling of lineages and cell types in the vertebrate brain. *Nature biotechnology*.

[132] Ramsköld, D., Luo, S., Wang, Y.-C., Li, R., Deng, Q., Faridani, O. R., Daniels, G. A., Khrebtukova, I., Loring, J. F., Laurent, L. C., *et al.* (2012). Full-length mrna-seq from single-cell levels of rna and individual circulating tumor cells. *Nature biotechnology*, **30**(8), 777.

[133] Raser, J. M. and O'shea, E. K. (2004). Control of stochasticity in eukaryotic gene expression. *science*, **304**(5678), 1811–1814.

[134] Regev, A., Teichmann, S., Lander, E. S., Amit, I., Benoist, C., Birney, E., Bodenmiller, B., Campbell, P., Carninci, P., Clatworthy, M., Clevers, H., Deplancke, B., Dunham, I., Eberwine, J., Eils, R., Enard, W., Farmer, A., Fugger, L., Gottgens, B., Hacohen, N., Haniffa, M., Hemberg, M., Kim, S. K., Klenerman, P., Kriegstein, A., Lein, E., Linnarsson, S., Lundeberg, J., Majumder, P., Marioni, J., Merad, M., Mhlanga, M., Nawijn, M., Netea, M., Nolan, G., Pe'er, D., Philipakis, A., Ponting, C. P., Quake, S. R., Reik, W., Rozenblatt-Rosen, O., Sanes, J. R., Satija, R., Shumacher, T., Shalek, A. K., Shapiro, E., Sharma, P., Shin, J., Stegle, O., Stratton, M., Stubbington, M. J. T., van Oudenaarden, A., Wagner, A., Watt, F. M., Weissman, J. S., Wold, B., Xavier, R. J., and Yosef, N. (2017). The human cell atlas. *bioRxiv*.

[135] Reyfman, P. A., Walter, J. M., Joshi, N., Anekalla, K. R., McQuattie-Pimentel, A. C., Chiu, S., Fernandez, R., Akbarpour, M., Chen, C.-I., Ren, Z., *et al.* (2018). Single-cell transcriptomic analysis of human lung provides insights into the pathobiology of pulmonary fibrosis. *American journal of respiratory and critical care medicine*, **0**(ja).

[136] Risso, D., Perraudeau, F., Gribkova, S., Dudoit, S., and Vert, J.-P. (2018). A general and flexible method for signal extraction from single-cell rna-seq data. *Nature communications*, **9**(1), 284.

[137] Rodda, L. B., Lu, E., Bennett, M. L., Sokol, C. L., Wang, X., Luther, S. A., Barres, B. A., Luster, A. D., Ye, C. J., and Cyster, J. G. (2018). Single-cell rna sequencing of lymph node stromal cells reveals niche-associated heterogeneity. *Immunity*, **48**(5), 1014–1028.

[138] Rosenberg, A. B., Roco, C. M., Muscat, R. A., Kuchina, A., Sample, P., Yao, Z., Graybuck, L. T., Peeler, D. J., Mukherjee, S., Chen, W., *et al.* (2018). Single-cell profiling of the developing mouse brain and spinal cord with split-pool barcoding. *Science*, **360**(6385), 176–182.

[139] Rostom, R., Svensson, V., Teichmann, S. A., and Kar, G. (2017). Computational approaches for interpreting scRNA-seq data. *FEBS Letters*.

[140] Roweis, S. T. and Saul, L. K. (2000). Nonlinear dimensionality reduction by locally linear embedding. *science*, **290**(5500), 2323–2326.

[141] Ryu, K. H., Huang, L., Kang, H. M., and Schiefelbein, J. (2019). Single-cell rna sequencing resolves molecular relationships among individual plant cells. *Plant physiology*, **179**(4), 1444–1456.

[142] Saelens, W., Cannoodt, R., Todorov, H., and Saeys, Y. (2018). A comparison of single-cell trajectory inference methods: towards more accurate and robust tools. *bioRxiv*, page 276907.

[143] Sasagawa, Y., Nikaido, I., Hayashi, T., Danno, H., Uno, K. D., Imai, T., and Ueda, H. R. (2013). Quartz-seq: a highly reproducible and sensitive single-cell rna sequencing method, reveals non-genetic gene-expression heterogeneity. *Genome biology*, **14**(4), 3097.

[144] Sasagawa, Y., Danno, H., Takada, H., Ebisawa, M., Tanaka, K., Hayashi, T., Kurisaki, A., and Nikaido, I. (2018). Quartz-seq2: a high-throughput single-cell rna-sequencing method that effectively uses limited sequence reads. *Genome biology*, **19**(1), 29.

[145] Schelker, M., Feau, S., Du, J., Ranu, N., Klipp, E., MacBeath, G., Schoeberl, B., and Raue, A. (2017). Estimation of immune cell content in tumour tissue using single-cell rna-seq data. *Nature communications*, **8**(1), 2032.

[146] Schiebinger, G., Shu, J., Tabaka, M., Cleary, B., Subramanian, V., Solomon, A., Gould, J., Liu, S., Lin, S., Berube, P., *et al.* (2019). Optimal-transport analysis of single-cell gene expression identifies developmental trajectories in reprogramming. *Cell*, **176**(4), 928–943.

[147] Schwarz, G. *et al.* (1978). Estimating the dimension of a model. *The annals of statistics*, **6**(2), 461–464.

[148] Sebé-Pedrós, A., Saudemont, B., Chomsky, E., Plessier, F., Mailhé, M.-P., Renno, J., Loe-Mie, Y., Lifshitz, A., Mukamel, Z., Schmutz, S., *et al.* (2018). Cnidarian cell type diversity and regulation revealed by whole-organism single-cell rna-seq. *Cell*, **173**(6), 1520–1534.

[149] Setty, M., Tadmor, M. D., Reich-Zeliger, S., Angel, O., Salame, T. M., Kathail, P., Choi, K., Bendall, S., Friedman, N., and Pe'er, D. (2016). Wishbone identifies bifurcating developmental trajectories from single-cell data. *Nature biotechnology*, **34**(6), 637–645.

[150] Shalek, A. K., Satija, R., Adiconis, X., Gertner, R. S., Gaublomme, J. T., Raychowdhury, R., Schwartz, S., Yosef, N., Malboeuf, C., Lu, D., *et al.* (2013). Single-cell transcriptomics reveals bimodality in expression and splicing in immune cells. *Nature*, **498**(7453), 236.

[151] Shalek, A. K., Satija, R., Shuga, J., Trombetta, J. J., Gennert, D., Lu, D., Chen, P., Gertner, R. S., Gaublomme, J. T., Yosef, N., *et al.* (2014). Single-cell rna-seq reveals dynamic paracrine control of cellular variation. *Nature*, **510**(7505), 363.

[152] Sharma, S., Wang, W., and Stolfi, A. (2019). Single-cell transcriptome profiling of the ciona larval brain. *Developmental biology*, **448**(2), 226–236.

[153] Shin, J., Berg, D. A., Zhu, Y., Shin, J. Y., Song, J., Bonaguidi, M. A., Enikolopov, G., Nauen, D. W., Christian, K. M., Ming, G.-l., *et al.* (2015). Single-cell rna-seq with waterfall reveals molecular cascades underlying adult neurogenesis. *Cell stem cell*, **17**(3), 360–372.

[154] Shnayder, M., Nachshon, A., Krishna, B., Poole, E., Boshkov, A., Binyamin, A., Maza, I., Sinclair, J., Schwartz, M., and Stern-Ginossar, N. (2018). Defining the transcriptional landscape during cytomegalovirus latency with single-cell rna sequencing. *MBio*, **9**(2), e00013–18.

[155] Shulse, C. N., Cole, B. J., Ciobanu, D., Lin, J., Yoshinaga, Y., Gouran, M., Turco, G. M., Zhu, Y., O'Malley, R. C., Brady, S. M., *et al.* (2019). High-throughput single-cell transcriptome profiling of plant cell types. *Cell reports*, **27**(7), 2241–2247.

[156] Skelly, D. A., Squiers, G. T., McLellan, M. A., Bolisetty, M. T., Robson, P., Rosenthal, N. A., and Pinto, A. R. (2018). Single-cell transcriptional profiling reveals cellular diversity and intercommunication in the mouse heart. *Cell reports*, **22**(3), 600–610.

[157] Spanjaard, B., Hu, B., Mitic, N., Olivares-Chauvet, P., Janjuha, S., Ninov, N., and Junker, J. P. (2018). Simultaneous lineage tracing and cell-type identification using crispr–cas9-induced genetic scars. *Nature biotechnology*, **36**(5), 469.

[158] Stephenson, W., Donlin, L. T., Butler, A., Rozo, C., Bracken, B., Rashidfarrokhi, A., Goodman, S. M., Ivashkiv, L. B., Bykerk, V. P., Orange, D. E., *et al.* (2018). Single-cell rna-seq of rheumatoid arthritis synovial tissue using low-cost microfluidic instrumentation. *Nature communications*, **9**(1), 791.

[159] Stevant, I., Neirijnck, Y., Borel, C., Escoffier, J., Smith, L. B., Antonarakis, S. E., Dermitzakis, E. T., and Nef, S. (2018). Deciphering cell lineage specification during male sex determination with single-cell rna sequencing. *Cell reports*, **22**(6), 1589–1599.

[160] Stoeckius, M., Hafemeister, C., Stephenson, W., Houck-Loomis, B., Chattopadhyay, P. K., Swerdlow, H., Satija, R., and Smibert, P. (2017). Simultaneous epitope and transcriptome measurement in single cells. *Nature methods*, **14**(9), 865.

[161] Street, K., Risso, D., Fletcher, R. B., Das, D., Ngai, J., Yosef, N., Purdom, E., and Dudoit, S. (2018). Slingshot: cell lineage and pseudotime inference for single-cell transcriptomics. *BMC Genomics*, **19**(1), 477.

[162] Svensson, V., Vento-Tormo, R., and Teichmann, S. A. (2018). Exponential scaling of single-cell rna-seq in the past decade. *Nature protocols*, **13**(4), 599.

[163] Tang, F., Barbacioru, C., Wang, Y., Nordman, E., Lee, C., Xu, N., Wang, X., Bodeau, J., Tuch, B. B., Siddiqui, A., *et al.* (2009). mrna-seq whole-transcriptome analysis of a single cell. *Nature methods*, **6**(5), 377.

[164] Tang, F., Barbacioru, C., Bao, S., Lee, C., Nordman, E., Wang, X., Lao, K., and Surani, M. A. (2010). Tracing the derivation of embryonic stem cells from the inner cell mass by single-cell rna-seq analysis. *Cell stem cell*, **6**(5), 468–478.

[165] Tang, F., Barbacioru, C., Nordman, E., Bao, S., Lee, C., Wang, X., Tuch, B. B., Heard, E., Lao, K., and Surani, M. A. (2011). Deterministic and stochastic allele specific gene expression in single mouse blastomeres. *PloS one*, **6**(6), e21208.

[166] Tang, J., Liu, J., Zhang, M., and Mei, Q. (2016). Visualizing large-scale and high-dimensional data. In *Proceedings of the 25th International Conference on World Wide Web*, pages 287–297. International World Wide Web Conferences Steering Committee.

[167] Tasic, B., Menon, V., Nguyen, T. N., Kim, T. K., Jarsky, T., Yao, Z., Levi, B., Gray, L. T., Sorensen, S. A., Dolbeare, T., *et al.* (2016). Adult mouse cortical cell taxonomy revealed by single cell transcriptomics. *Nature neuroscience*, **19**(2), 335.

[168] Tibshirani, R., Walther, G., and Hastie, T. (2001). Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **63**(2), 411–423.

[169] Tikhonova, A. N., Dolgalev, I., Hu, H., Sivaraj, K. K., Hoxha, E., Cuesta-Domínguez, Á., Pinho, S., Akhmetzyanova, I., Gao, J., Witkowski, M., *et al.* (2019). The bone marrow microenvironment at single-cell resolution. *Nature*, **569**(7755), 222.

[170] Tiklová, K., Björklund, Å. K., Lahti, L., Fiorenzano, A., Nolbrant, S., Gillberg, L., Volakakis, N., Yokota, C., Hilscher, M. M., Hauling, T., *et al.* (2019). Single-cell rna sequencing reveals midbrain dopamine neuron diversity emerging during mouse brain development. *Nature communications*, **10**(1), 581.

[171] Tirosh, I., Izar, B., Prakadan, S. M., Wadsworth, M. H., Treacy, D., Trombetta, J. J., Rotem, A., Rodman, C., Lian, C., Murphy, G., *et al.* (2016). Dissecting the multicellular ecosystem of metastatic melanoma by single-cell rna-seq. *Science*, **352**(6282), 189–196.

[172] Torre, E., Dueck, H., Shaffer, S., Gospocic, J., Gupte, R., Bonasio, R., Kim, J., Murray, J., and Raj, A. (2018). Rare cell detection by single-cell rna sequencing as guided by single-molecule rna fish. *Cell systems*, **6**(2), 171–179.

[173] Trapnell, C., Cacchiarelli, D., Grimsby, J., Pokharel, P., Li, S., Morse, M., Lennon, N. J., Livak, K. J., Mikkelsen, T. S., and Rinn, J. L. (2014). Pseudo-temporal ordering of individual cells reveals dynamics and regulators of cell fate decisions. *Nature biotechnology*, **32**(4), 381.

[174] Treutlein, B., Brownfield, D. G., Wu, A. R., Neff, N. F., Mantalas, G. L., Espinoza, F. H., Desai, T. J., Krasnow, M. A., and Quake, S. R. (2014). Reconstructing lineage hierarchies of the distal lung epithelium using single-cell rna-seq. *Nature*, **509**(7500), 371.

[175] Treutlein, B., Lee, Q. Y., Camp, J. G., Mall, M., Koh, W., Shariati, S. A. M., Sim, S., Neff, N. F., Skotheim, J. M., Wernig, M., *et al.* (2016). Dissecting direct reprogramming from fibroblast to neuron using single-cell rna-seq. *Nature*.

[176] Tsuyuzaki, K., Sato, H., Sato, K., and Nikaido, I. (2019). Benchmarking principal component analysis for large-scale single-cell rna-sequencing. *bioRxiv*, page 642595.

[177] Vallejos, C. A., Marioni, J. C., and Richardson, S. (2015). Basics: Bayesian analysis of single-cell sequencing data. *PLoS computational biology*, **11**(6), e1004333.

[178] van der Wijst, M. G., Brugge, H., de Vries, D. H., Deelen, P., Swertz, M. A., and Franke, L. (2018). Single-cell rna sequencing identifies celltype-specific cis-eqtls and co-expression qtls. *Nature genetics*, **50**(4), 493.

[179] Van Dijk, D., Sharma, R., Nainys, J., Yim, K., Kathail, P., Carr, A. J., Burdziak, C., Moon, K. R., Chaffer, C. L., Pattabiraman, D., *et al.* (2018). Recovering gene interactions from single-cell data using data diffusion. *Cell*, **174**(3), 716–729.

[180] van Galen, P., Hovestadt, V., Wadsworth II, M. H., Hughes, T. K., Griffin, G. K., Battaglia, S., Verga, J. A., Stephansky, J., Pastika, T. J., Story, J. L., *et al.* (2019). Single-cell rna-seq reveals aml hierarchies relevant to disease progression and immunity. *Cell*, **176**(6), 1265–1281.

[181] van Unen, V., Li, N., Molendijk, I., Temurhan, M., Höllt, T., van der Meulen-de, A. E., Verspaget, H. W., Mearin, M. L., Mulder, C. J., van Bergen, J., *et al.* (2016). Mass cytometry of the human mucosal immune system identifies tissue-and disease-associated immune subsets. *Immunity*, **44**(5), 1227–1239.

[182] Venteicher, A. S., Tirosh, I., Hebert, C., Yizhak, K., Neftel, C., Filbin, M. G., Hovestadt, V., Escalante, L. E., Shaw, M. L., Rodman, C., *et al.* (2017). Decoupling genetics, lineages, and microenvironment in idh-mutant gliomas by single-cell rna-seq. *Science*, **355**(6332), eaai8478.

[183] Vickovic, S., Ståhl, P. L., Salmén, F., Giatrellis, S., Westholm, J. O., Mollbrink, A., Navarro, J. F., Custodio, J., Bienko, M., Sutton, L.-A., *et al.* (2016). Massive and parallel expression profiling using microarrayed single-cell sequencing. *Nature communications*, **7**, 13182.

[184] Villani, A.-C., Satija, R., Reynolds, G., Sarkizova, S., Shekhar, K., Fletcher, J., Griesbeck, M., Butler, A., Zheng, S., Lazo, S., *et al.* (2017). Single-cell rna-seq reveals new types of human blood dendritic cells, monocytes, and progenitors. *Science*, **356**(6335), eaah4573.

[185] Wagner, D. E., Weinreb, C., Collins, Z. M., Briggs, J. A., Megason, S. G., and Klein, A. M. (2018). Single-cell mapping of gene expression landscapes and lineage in the zebrafish embryo. *Science*, **360**(6392), 981–987.

[186] Wagner, F., Barkley, D., and Yanai, I. (2019). Enhance: Accurate denoising of single-cell rna-seq data. *BioRxiv*, page 655365.

[187] Wan, S., Kim, J., and Won, K. J. (2018). Sharp: Single-cell rna-seq hyper-fast and accurate processing via ensemble random projection. *bioRxiv*, page 461640.

[188] Wang, B., Ramazzotti, D., De Sano, L., Zhu, J., Pierson, E., and Batzoglou, S. (2017). Simlr: a tool for large-scale single-cell analysis by multi-kernel learning. *bioRxiv*, page 118901.

[189] Wang, D. and Gu, J. (2018). Vasc: Dimension reduction and visualization of single-cell rna-seq data by deep variational autoencoder. *Genomics, proteomics & bioinformatics*, **16**(5), 320–331.

[190] Wang, X., Allen, W. E., Wright, M. A., Sylwestrak, E. L., Samusik, N., Vesuna, S., Evans, K., Liu, C., Ramakrishnan, C., Liu, J., *et al.* (2018). Three-dimensional intact-tissue sequencing of single-cell transcriptional states. *Science*, **361**(6400), eaat5691.

[191] Welch, J. D., Hartemink, A. J., and Prins, J. F. (2016). Slicer: inferring branched, nonlinear cellular trajectories from single cell rna-seq data. *Genome biology*, **17**(1), 106.

[192] Wen, T., Aronow, B. J., Rochman, Y., Rochman, M., Kiran, K., Dexheimer, P. J., Putnam, P., Mukkada, V., Foote, H., Rehn, K., *et al.* (2019). Single-cell rna sequencing identifies inflammatory tissue t cells in eosinophilic esophagitis. *The Journal of clinical investigation*, **129**(5).

[193] Wolf, F. A., Hamey, F., Plass, M., Solana, J., Dahlin, J. S., Gottgens, B., Rajewsky, N., Simon, L., and Theis, F. J. (2018a). Graph abstraction reconciles clustering with trajectory inference through a topology preserving map of single cells. *bioRxiv*, page 208819.

[194] Wolf, F. A., Angerer, P., and Theis, F. J. (2018b). Scanpy: large-scale single-cell gene expression data analysis. *Genome biology*, **19**(1), 15.

[195] Wu, A. R., Neff, N. F., Kalisky, T., Dalerba, P., Treutlein, B., Rothenberg, M. E., Mburu, F. M., Mantalas, G. L., Sim, S., Clarke, M. F., *et al.* (2014). Quantitative assessment of single-cell rna-sequencing methods. *Nature methods*, **11**(1), 41.

[196] Wu, Y., Tamayo, P., and Zhang, K. (2018). Visualizing and interpreting single-cell gene expression datasets with similarity weighted nonnegative embedding. *Cell systems*, **7**(6), 656–666.

[197] Wu, Y. E., Pan, L., Zuo, Y., Li, X., and Hong, W. (2017). Detecting activated cell populations using single-cell rna-seq. *Neuron*, **96**(2), 313–329.

[198] Yan, L., Yang, M., Guo, H., Yang, L., Wu, J., Li, R., Liu, P., Lian, Y., Zheng, X., Yan, J., *et al.* (2013). Single-cell rna-seq profiling of human preimplantation embryos and embryonic stem cells. *Nature structural & molecular biology*, **20**(9), 1131.

[199] Yuan, J. and Sims, P. A. (2016). An automated microwell platform for large-scale single cell rna-seq. *Scientific reports*, **6**, 33883.

[200] Zappia, L., Phipson, B., and Oshlack, A. (2017). Splatter: simulation of single-cell rna sequencing data. *Genome biology*, **18**(1), 174.

[201] Zappia, L., Phipson, B., and Oshlack, A. (2018). Exploring the single-cell rna-seq analysis landscape with the scrna-tools database. *PLoS computational biology*, **14**(6), e1006245.

[202] Zeisel, A., Muñoz-Manchado, A. B., Codeluppi, S., Lönnerberg, P., La Manno, G., Juréus, A., Marques, S., Munguba, H., He, L., Betsholtz, C., *et al.* (2015). Cell types in the mouse cortex and hippocampus revealed by single-cell rna-seq. *Science*, **347**(6226), 1138–1142.

[203] Zeisel, A., Hochgerner, H., Lönnerberg, P., Johnsson, A., Memic, F., Van Der Zwan, J., Häring, M., Braun, E., Borm, L. E., La Manno, G., *et al.* (2018). Molecular architecture of the mouse nervous system. *Cell*, **174**(4), 999–1014.

[204] Zhang, H. and Miller, R. H. (1995). Asynchronous differentiation of clonally related spinal cord oligodendrocytes. *Molecular and Cellular Neuroscience*, **6**(1), 16–31.

[205] Zheng, G. X., Terry, J. M., Belgrader, P., Ryvkin, P., Bent, Z. W., Wilson, R., Ziraldo, S. B., Wheeler, T. D., McDermott, G. P., Zhu, J., *et al.* (2017). Massively parallel digital transcriptional profiling of single cells. *Nature communications*, **8**, 14049.

[206] Zhong, S., Zhang, S., Fan, X., Wu, Q., Yan, L., Dong, J., Zhang, H., Li, L., Sun, L., Pan, N., *et al.* (2018). A single-cell rna-seq survey of the developmental landscape of the human prefrontal cortex. *Nature*, **555**(7697), 524.

[207] Ziegenhain, C., Vieth, B., Parekh, S., Reinius, B., Guillaumet-Adkins, A., Smets, M., Leonhardt, H., Heyn, H., Hellmann, I., and Enard, W. (2017). Comparative analysis of single-cell rna sequencing methods. *Molecular cell*, **65**(4), 631–643.

[208] Zou, H. and Xue, L. (2018). A selective overview of sparse principal component analysis. *Proceedings of the IEEE*, **106**(8), 1311–1320.