A comparison of single-cell trajectory inference methods: towards more accurate and robust tools

05 March, 2018

Wouter Saelens* [1 2], Robrecht Cannoodt* [1 2 3], Helena Todorov[1 2 4], Yvan Saeys[1 2]

* Equal contribution

[1] Data mining and Modelling for Biomedicine, VIB Center for Inflammation Research, Ghent, Belgium.

[2] Department of Applied Mathematics, Computer Science and Statistics, Ghent University, Ghent, Belgium.

[3] Center for Medical Genetics, Ghent University Hospital, Ghent, Belgium.

[4] Centre International de Recherche en Infectiologie, Inserm, U1111, Université Claude Bernard Lyon 1, CNRS, UMR5308, École Normale Supérieure de Lyon, Univ Lyon, F-69007, Lyon, France

Abstract

Using single-cell -omics data, it is now possible to computationally order cells along trajectories, allowing the unbiased study of cellular dynamic processes. Since 2014, more than 50 trajectory inference methods have been developed, each with its own set of methodological characteristics. As a result, choosing a method to infer trajectories is often challenging, since a comprehensive assessment of the performance and robustness of each method is still lacking. In order to facilitate the comparison of the results of these methods to each other and to a gold standard, we developed a global framework to benchmark trajectory inference tools. Using this framework, we compared the trajectories from a total of 29 trajectory inference methods, on a large collection of real and synthetic datasets. We evaluate methods using several metrics, including accuracy of the inferred ordering, correctness of the network topology, code quality and user friendliness. We found that some methods, including Slingshot, TSCAN and Monocle DDRTree, clearly outperform other methods, although their performance depended on the type of trajectory present in the data. Based on our benchmarking results, we therefore developed a set of guidelines for method users. However, our analysis also indicated that there is still a lot of room for improvement, especially for methods detecting complex trajectory topologies. Our evaluation pipeline can therefore be used to spearhead the development of new scalable and more accurate methods, and is available at github.com/dynverse/dynverse.

To our knowledge, this is the first comprehensive assessment of trajectory inference methods. For now, we exclusively evaluated the methods on their default parameters, but plan to add a detailed parameter tuning procedure in the future. We gladly welcome any discussion and feedback on key decisions made as part of this study, including the metrics used in the benchmark, the quality control checklist, and the implementation of the method wrappers. These discussions can be held at github.com/dynverse/dynverse/issues.

Introduction

Single-cell -omics technologies now make it possible to model biological systems more accurately than ever before[1]. One area where single-cell data has been particularly useful is in the study of cellular dynamic processes, such as the cell cycle, cell differentiation and cell activation[2]. Such dynamic processes can be computationally modelled using trajectory inference (TI) methods (also known as pseudemporal ordering methods), which use single-cell profiles from a population in which the cells are at different unknown points in the dynamic process[3,4,5]. These methods computationally order the cells along a trajectory topology, which can be linear, bifurcating, or a more complex tree or graph structure. Because TI methods offer an unbiased and transcriptome-wide understanding of a dynamic process[1], they allow the objective identification of new (primed) subsets of cells[6], delineation of a differentiation tree[7,8] and inference of regulatory interaction responsible for one or more bifurcations[9]. Current applications of TI focus on specific subsets of cells, but ongoing efforts to construct transcriptomic catalogues of whole organisms[10,11] underline the urgency for accurate, scalable[9,12] and user-friendly TI methods.

Table 1 Overview of the trajectory inference methods included in this study, and several characteristics thereof. This table will be continuously updated online.

Method	Date	Most complex trajectory type	Fixes topology	Prior required	Prior optional	Evaluated	Reference
Monocle ICA	01/04/2014	Tree	Parameter	# branches	None	Yes	[13]
Wanderlust	24/04/2014	Linear	Fixed	Start cell(s)	None	Yes	[14]
SCUBA	30/12/2014	Tree	Free	None	Time course, Marker genes	Yes	[15]
Sincell	27/01/2015	Tree	Free	None	None	Yes	[16]
NBOR	08/06/2015	Linear	TBD	TBD	TBD	No[a i]	[6]
Waterfall	03/09/2015	Linear	Fixed	None	None	Yes	[17]
gpseudotime	15/09/2015	Linear	TBD	TBD	TBD	No[c]	[18]
Embeddr	18/09/2015	Linear	Fixed	None	None	Yes	[19]
ECLAIR	12/01/2016	Tree	TBD	TBD	TBD	No[f]	[20]
DPT	08/02/2016	Bifurcation	Fixed	None	Marker genes	Yes	[21]
Pseudogp	05/04/2016	Linear	Fixed	None	None	Yes	[22]
SLICER	09/04/2016	Graph	Free	Start cell(s)	End cell(s), Marker genes	Yes	[23]
SCell	19/04/2016	Linear	TBD	TBD	TBD	No[e]	[24]
Wishbone	02/05/2016	Bifurcation	Parameter	Start cell(s), # end states	Marker genes	Yes	[25]
TSCAN	13/05/2016	Tree	Free	None	None	Yes	[26]
SCOUP	08/06/2016	Multifurcation	Parameter	Start cell(s), Cell grouping, # end states	None	Yes	[27]
DeLorean	17/06/2016	Linear	TBD	TBD	TBD	No[g]	[28]
StemID	21/06/2016	Tree	Free	None	None	Yes	[29]
Ouija	23/06/2016	Linear	Fixed	Marker genes	None	Yes	[30]
Mpath	30/06/2016	Tree	Free	Cell grouping	None	Yes	[31]
cellTree	13/08/2016	Tree	Free	None	Cell grouping	Yes	[32]
WaveCrest	17/08/2016	Linear	TBD	Time course	None	No[f]	[33]
SCIMITAR	04/10/2016	Linear	Fixed	None	None	Yes	[34]
SCORPIUS	07/10/2016	Linear	Fixed	None	None	Yes	[35]
SCENT	30/10/2016	Linear	TBD	TBD	TBD	No[d]	[36]
k-branches	15/12/2016	Tree	TBD	TBD	TBD	No[h]	[37]
SLICE	19/12/2016	Tree	Free	None	Cell grouping, Marker genes	Yes	[38]
Topslam	13/02/2017	Linear	Fixed	Start cell(s)	None	Yes	[39]
Monocle DDRTree	21/02/2017	Tree	Free	None	# end states	Yes	[40]
Granatum	22/02/2017	Tree	TBD	TBD	TBD	No[e]	[41]
GPfates	03/03/2017	Multifurcation	Parameter	# end states	None	Yes	[42]
MFA	15/03/2017	Multifurcation	Parameter	# end states	None	Yes	[43]
PHATE	24/03/2017	Tree	TBD	TBD	TBD	No[h]	[44]
TASIC	04/04/2017	Tree	TBD	TBD	TBD	No[a e]	[45]
SOMSC	05/04/2017	Tree	TBD	TBD	TBD	No[a]	[46]
Slingshot	19/04/2017	Tree	Free	None	Start cell(s), End cell(s)	Yes	[47]
scTDA	01/05/2017	Linear	TBD	TBD	TBD	No[f]	[48]
UNCURL	31/05/2017	Linear	TBD	TBD	TBD	No[f]	[49]
reCAT	19/06/2017	Cycle	Fixed	None	None	Yes	[50]
FORKS	20/06/2017	Tree	TBD	Start cell(s)	None	No[f j]	[51]
MATCHER	24/06/2017	Linear	TBD	TBD	TBD	No[j]	[52]
PhenoPath	06/07/2017	Linear	Fixed	None	None	Yes	[53]
HopLand	12/07/2017	Linear	TBD	TBD	TBD	No[a j]	[54]
SoptSC	26/07/2017	Linear	TBD	Start cell(s)	None	No[a j]	[55]
PBA	30/07/2017	Multifurcation	TBD	TBD	TBD	No[j]	[56]
BGP	01/08/2017	Bifurcation	TBD	TBD	TBD	No[j]	[57]
scanpy	09/08/2017	Bifurcation	TBD	TBD	TBD	No[j]	[58]
B-RGPs	01/09/2017	Acyclic graph	TBD	TBD	TBD	No[j]	[59]
WADDINGTON-OT	27/09/2017	Graph	TBD	TBD	TBD	No[b j]	[60]
AGA	27/10/2017	Disconnected graph	TBD	TBD	TBD	No[j]	[61]
GPseudoRank	30/10/2017	Linear	TBD	TBD	TBD	No[a j]	[62]
p-Creode	15/11/2017	Tree	TBD	TBD	TBD	No[j]	[63]
iCpSc	30/11/2017	Linear	TBD	TBD	TBD	No[d j]	[64]
GrandPrix	03/12/2017	Multifurcation	TBD	Time course	None	No[j]	[65]
Topographer	21/01/2018	Tree	TBD	None	Start cell(s)	No[j]	[66]
CALISTA	31/01/2018	Graph	TBD	None	None	No[j]	[67]
scEpath	05/02/2018	Tree	TBD	TBD	TBD	No[a j]	[68]
MERLoT	08/02/2018	Tree	TBD	TBD	TBD	No[j]	[69]
ElPiGraph.R	04/03/2018	Graph	TBD	TBD	TBD	No[j]	

[a] Not free
[b] Unavailable
[c] Superseded by another method
[d] Requires data types other than expression
[e] No programming interface
[f] Unresolved errors during wrapping
[g] Too slow (requires more than one hour on a 100x100 dataset)
[h] Doesn't return an ordering
[i] Requires additional user input during the algorithm (not prior information)
[j] Published later than 2017-05-01 to be included in the current version of the evaluation

A plethora of TI methods has been developed over the last years, and even more are being created every month (**Supplementary Figure 1a**). It is perhaps surprising that of the 59 methods in existence today, almost all methods have a unique combination of characteristics (**Table 1**), in terms of the required inputs (prior information), produced outputs (topology fixing and trajectory type) and methodology used (not shown). One distinctive characteristic of TI methods is whether the topology of the trajectory is inferred computationally, or was fixed by design. Early TI methods typically fixed the topology algorithmically (e.g. linear[14,6,17,18] or bifurcating[21,25]), or through parameters provided by the user[13,27]. These methods therefore mainly focused on correctly ordering the cells along this fixed topology. Other methods attempt to infer the topology computationally, which increases the difficulty of the problem at hand, but allows these methods to be broadly applicable on more use cases. Methods that perform topology inference are still in the minority, though current trends suggest this will soon change (**Supplementary Figure 1c**). A particularly interesting development is presented in the AGA method[61] which is the only TI method currently able to deal with disconnected graphs.

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

A reductionist approach to characterising TI methods consists in dissecting them into a set of algorithmic components, as any component can have a significant impact on the performance, scalability, and output data structures. Across all TI methods, these components can be broadly grouped into two stages; (i) conversion to a simplified representation using dimensionality reduction, clustering or graph building and (ii) ordering the cells along the simplified representation[4]. Interestingly, components are frequently shared between different algorithms (**Supplementary Figure 2**). For example, minimal spanning trees (MST), used in the first single-cell RNA-seq trajectory inference methods[13], is shared by almost half of the methods we evaluated (**Supplementary Figure 2b**).

Given the diversity in TI methods, an important issue to address is a quantitative assessment of the performance and robustness of the existing TI methods. Many attempts at tackling this issue have already been made[21,26,23,27,32,35,42,43], but due to the high number of TI methods available today and the great diversity in the outputted data structures, a comprehensive benchmarking evaluation of TI methods is still lacking. This is problematic, as new users to the field are confronted with a wide array of TI methods, without a clear idea about what method would be the most optimal for their problem. Moreover, the strengths and weaknesses of existing methods need to be assessed, so that new developments in the field can focus on improving the current state-of-the-art.

Results

In this study, we performed a comprehensive evaluation for 29 TI methods (Overview: **Figure 1a**, Extended overview: **Supplementary Figure 3**). The inclusion criterion for TI methods was primarily based on their free availability, the presence of a programming interface, and the date of publication (**Table 1**). Only methods published before june 2017 are included in the current version of the evaluation, while more recent methods will be added in the next version. The evaluation comprised three core aspects: (i) source-code and literature-based characterisation of TI methods, (ii) assessment of the accuracy and scalability of TI methods by comparing predicted trajectories with a gold standard, and (iii) a quality control of the provided software and documentation.

In order to make gold standard trajectories and predicted trajectories directly comparable to one another, we developed a common probabilistic model for representing trajectories from all possible sources (**Figure 1b**). In this model, the overall topology is represented by a network of "milestones", and the cells are placed within the space formed by each set of connected milestones. We defined a set of metrics for comparing the likeness of such trajectories, each assessing a different aspect of the trajectory: the similarity in cell ordering, the cell neighbourhood and the topology (**Figure 1c**). The data compendium consisted of both synthetic datasets, which offer the most exact gold standard, and real datasets, which offer the highest biological relevance. These real datasets came from a variety of single-cell technologies, organisms, and dynamic processes, and contain several types of trajectory topologies (**Supplementary Figure 4** and **Supplementary Table 1**). To generate synthetic datasets, we simulated a gene regulatory network using a thermodynamic model of gene regulation[70], and subsequently simulated a

single-cell profiling experiment by matching the distributions of the synthetic data with real reference datasets (**Supplementary Figure 5**).

As the aim of this study is to provide both guidelines for the use of TI methods and a base for the development of new TI methods, we not only assessed the accuracy of the predictions made by a trajectory, but also investigated the quality of the method's implementation. To do this, we scored each method using a checklist of important scientific and software development practices, including software packaging, documentation, automated code testing, and peer review. This allowed us to rank each method based on its user friendliness, developer friendliness, and potential broad applicability on new unseen datasets. Finally, using both the benchmark results and quality control, we produced a flow chart with practical guidelines for selecting the most appropriate TI method for a given use case.

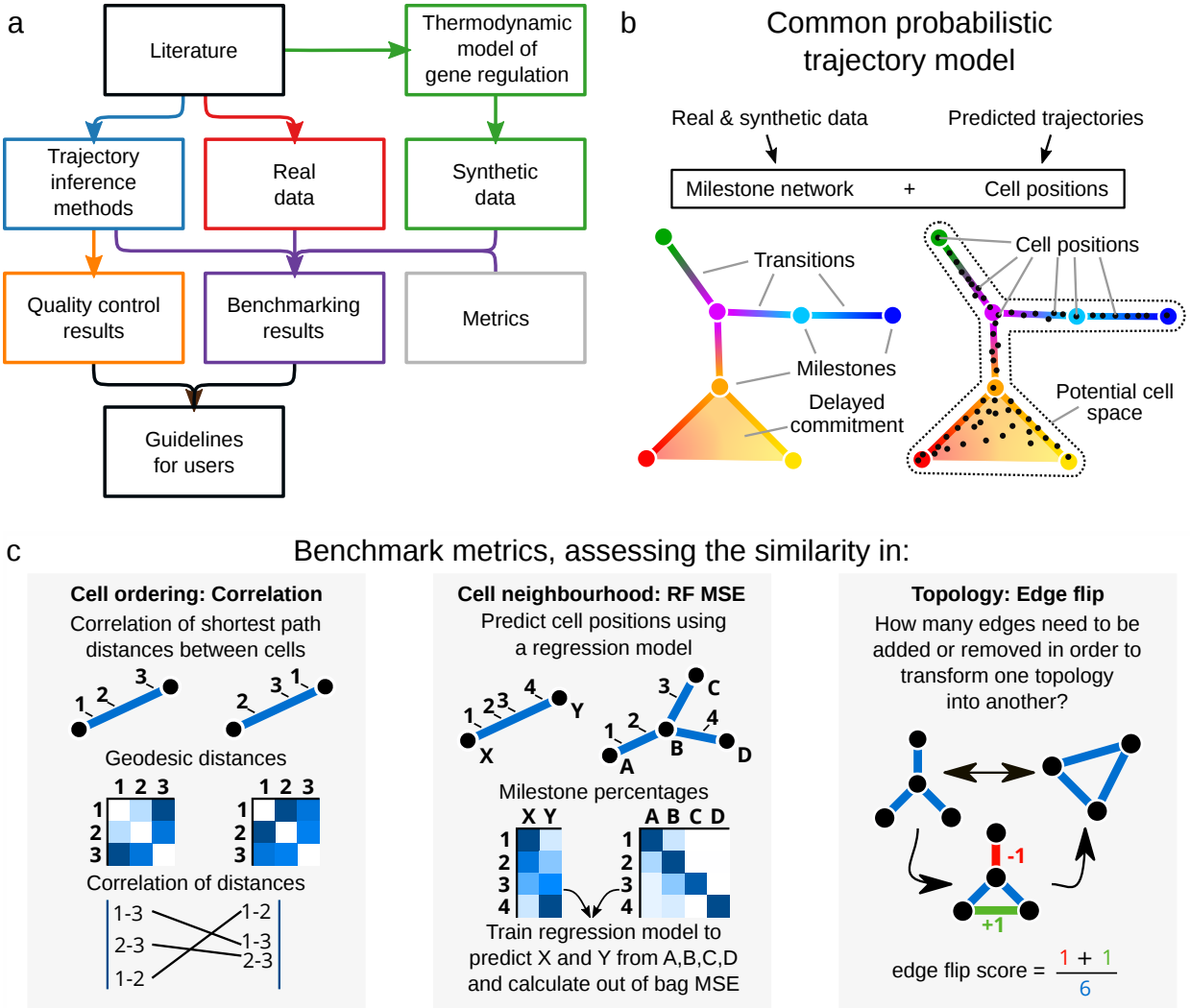


Figure 1 An overview of several key aspects of the evaluation. a) A schematic overview of our evaluation pipeline. b) In order to make any two trajectories comparable with one another, a common trajectory model was used to represent gold standard trajectories from the real and synthetic datasets, as well as any predicted trajectories from TI methods. c) Three metrics were defined in order to assess the similarity in cell ordering, cell neighbourhood, and topology, for any two trajectories.

Evaluation of trajectory inference methods

An overview of the main results from this study is shown in **Figure 2**. This includes an overview of the results obtained from the method characterisation (**Figure 2a**), the benchmarking evaluation (**Figure 2b**), and the quality control evaluation (**Figure 2c**).

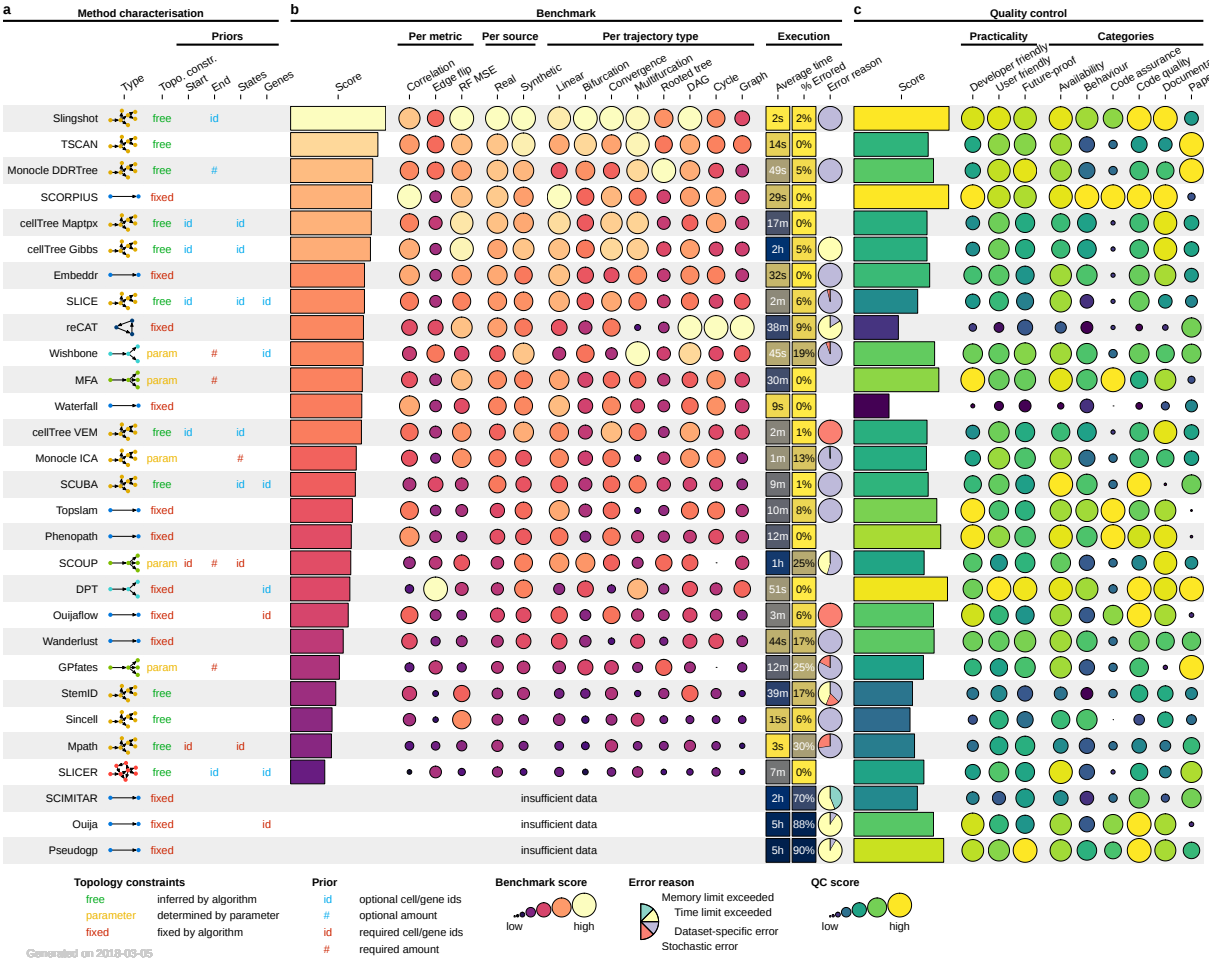


Figure 2 Overview of the results on the evaluation of 29 TI methods. a) The methods were characterised according to the most complex trajectory type they can infer, whether or not the inferred topology is constrained by the algorithm or a parameter, and which prior information a method requires or can optionally use. b) The methods are ordered according to the overall score achieved in the benchmark evaluation. Also shown are the aggregated scores per metric, source and trajectory type, as well as the average execution time across all datasets and the percentage of executions in which no output was produced. c) Overall performance in the quality control evaluation is highly variable, even amongst the highest ranked methods according to the benchmark evaluation. Also listed are the quality control scores aggregated according to practicality and the different categories.

Having ordered all methods by their overall benchmarking score, we found that Slingshot predicted the most accurate trajectories, followed by TSCAN and Monocle DDRTree. When we looked at the benchmark scores per trajectory type, Slingshot was the only method that performed well across most trajectory types. However, we found that several methods were specialised in predicting specific trajectory types; for example SCORPIUS for linear trajectories, reCAT for cycles, and Monocle DDRTree for trees.

We observed a high correlation (0.7-0.9) between results originating from real datasets versus those originating from synthetic datasets (**Supplementary Figure 6**). This confirms both the relevance of the synthetic data and the accuracy of the gold stan-

dard in the real datasets. However, this correlation was lower for converging and multifurcating datasets, potentially because only a small number of real datasets were available for such topologies (**Supplementary Table 1**).

As the different metrics were selected to assess the correctness of a trajectory in various approaches, it is expected to observe differences in rankings of TI methods amongst the different metrics. While we saw no direct link between the edge flip scores versus the correlation or RF MSE metric, methods that obtained a high correlation score also tended to obtain higher RF MSE scores (**Supplementary Figure 7**). In addition, the methods that were able to detect more complex trajectory types also obtained higher edge flip scores, in comparison to methods whose trajectory topology was fixed to a simple trajectory topology. We will explore this issue in more detail in a further section.

During the execution of a method on a dataset, if the time limit (>6h) or memory limit (32GB) was exceeded, or an error was produced, a zero score was returned for that execution. If a method consistently generated errors on this dataset across all replicates, the error is called “data-specific”, otherwise “stochastic”. Several methods obtained a high overall score despite having 5 to 10% of failed executions (e.g. Monocle DDRTree, SLICE, Wishbone), meaning these methods could rank even higher if not for the failed executions. Methods that do not scale well with respect to the number of cells or genes will exceed the time or memory limits for the largest datasets (reCAT, SCOUP, StemID). For a few methods, the time or memory limits were exceeded too often, making the benchmarking results uncomparable to those of other methods (SCIMITAR, Ouija, Pseudogp).

Trajectory types and topology complexity

In most cases, the methods which were specifically designed to handle a particular trajectory type, also performed better on data containing this particular trajectory type (**Supplementary Figure 8**). These methods had typically better edge flip scores - as can be expected - and RF MSE scores, compared to methods not able to handle the particular trajectory type. However, the correlation score typically followed the opposite pattern, where methods restricted to linear trajectory types, such as embeddr, SCORPIUS and phenopath, produced the best ordering, irrespective of whether the dataset contained a linear trajectory or more complex trajectories. To further investigate the effect of trajectory complexity on performance, we divided them in two groups: linear methods (restricted to linear and cyclic topologies) and non-linear methods. While there were no significant differences in performance on linear datasets, non-linear methods had significantly higher edge flip scores but lower correlation scores on datasets containing more complex trajectory types (**Supplementary Figure 9**). Together, this indicates that current non-linear methods potentially contain less accurate ordering algorithms. A combination of the ordering methods from the top linear methods, combined with the topology inference from top non-linear methods, could therefore be a possibility for future research.

Despite their similar overall performance, the topologies predicted by the top methods differed considerably. Trajectories detected using the default parameters of slingshot and cellTree tended to be simpler, while those detected by TSCAN and Monocle DDRTree gravitated towards more complex topologies (**Figure 3**). Monocle DDRTree, for example frequently predicted a tree topology, even when only a cyclic, linear or bifurcating topology was present in the data (**Figure 3a**). Trajectories generated by Monocle DDRTree (at default parameters) tended to contain more nodes and edges (**Figure 3b**), which could give this method an advantage on datasets with complex tree topologies, but could also explain its relatively low performance on linear and bifurcating datasets. Indeed, when we assessed how often a method can infer the correct topology, slingshot and TSCAN were good at correctly predicting linear, bifurcating and converging trajectories, while Monocle DDRTree was by far the best method to infer tree topologies. Nonetheless, the overall accuracy of topology prediction was very low, with Slingshot correctly predicting bifurcating and converging topologies for half of the datasets, and Monocle DDRTree predicting the correct tree topology in 12% of the cases (**Supplementary Figure 10**). Inferring the correct topology without parameter tuning, is therefore still an open challenge. Conversely, when the data contains a complex trajectory structure, TI will currently still require a considerable guidance by the end user, to either optimise the parameters or to choose the method with which the output best fits the user’s

expert knowledge.

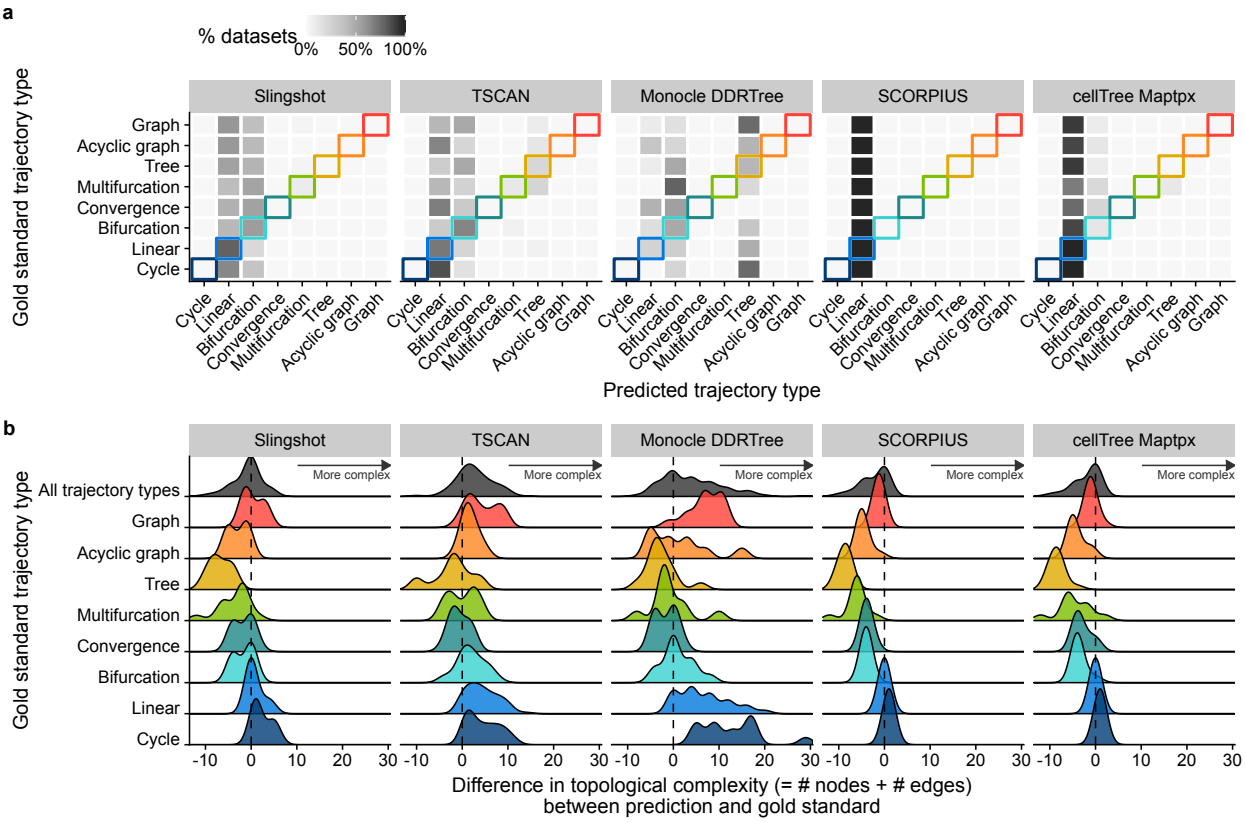


Figure 3 Comparing the ability of the top TI methods to detect the correct trajectory type. a) % of datasets on which a TI method detects a particular trajectory type, compared with the correct trajectory type. b) Distributions of the topological complexity, defined by the sum of the number of milestones and edges between milestones, compared with the true trajectory type present in the data.

Effect of prior information

In the current version of the evaluation, we only provided prior information when a method required it . We did not observe a major difference in method performance between methods which did and did not receive prior information (**Supplementary Figure 11**). Rather, methods which received prior information were on average positioned in the middle of the ranking. Furthermore, we could not find any dataset where methods which received prior information performed significantly better than other methods.

Algorithm components

We assessed whether the components of an algorithm could be predictive of a method's performance, using both random forest classification and statistical testing. Methods which included principal curves (such as Slingshot and SCORPIUS), k-means (which include Slingshot and several other top scoring methods) and some graph building (which include almost every top scoring method) tended to have a slightly higher performance (**Supplementary Figure 12** and **Supplementary Table 2**). On the other hand, methods using t-SNE and ICA for their dimensionality reduction were ranked lower, although this was not statistically significant.

Method quality control

While not directly related to the accuracy of the inferred trajectory, the quality of the implementation is also an important evaluation metric. Good unit testing assures that the implementation is correct, good documentation makes it easier for potential users to apply the method on their data, and overall good code quality makes it possible for other developers to adapt the method and extend it further. We therefore looked at the implementation of each method, and assessed its quality using a transparent scoring scheme (**Supplementary Table 3**). The individual quality checks can be grouped in two ways: what aspect of the method they investigate (availability, code quality, code assurance, documentation, method's behaviour at runtime and the depth by which the method was presented in its study) or which purpose(s) they serve (user friendliness, developer friendliness or future proof). These categorisations can help current developers to improve their tool, and guide the selection of users and developers to use these tools for their purpose. After publishing this preprint, we will contact the authors of each method, allowing them to improve their method before the final publishing of the evaluation.

We found that most methods fulfilled the basic criteria, such as free availability and basic code quality criteria (**Figure 4**). We found that recent methods had a slightly better quality than older methods (**Supplementary Figure 13**). However, several quality aspects were consistently lacking for the majority of the methods (**Figure 4 right**) and we believe that these should receive extra attention from developers. Although these outstanding issues cover all five categories, code assurance and documenta-

tion in particular are problematic areas, notwithstanding several studies pinpointing these as good practices[71,72].

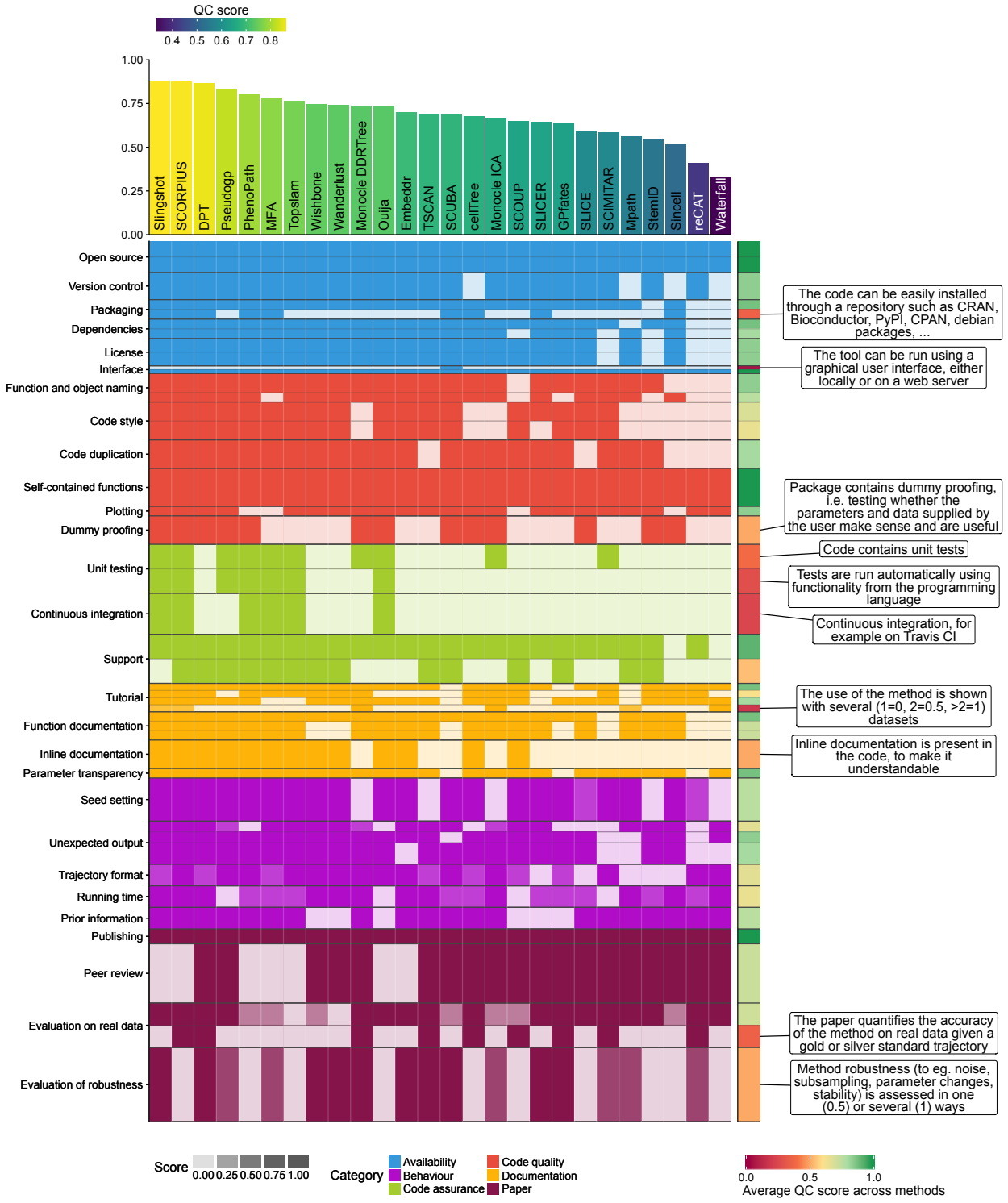


Figure 4 Overview of the quality control results for every method. Shown is the score given for each methods on every item from our quality control score sheet (**Supplementary Table 3**). Each aspect of the quality control was part of a category, and each category was weighted so that it contributed equally to the final quality score. Within each category, each aspect also received a weight depending on how often it was mentioned in a set of papers discussing good practices in tool development and evaluation. This weight is represented in the plot as distance on the y-axis. Top: Average QC score for each method. Right:

The average score of each quality control item. Shown into more detail are those items which had a score lower than 0.5.

We observed no clear relation between method quality and method performance (**Figure 2** and **Supplementary Figure 14**). We could also not find any quality aspect which was significantly associated with method performance.

Practical guidelines

Based on the results of our benchmark, we created a set of practical guidelines for method users **Figure 5**. As a method's performance is heavily dependent on the trajectory type being studied, the choice of method will by primarily driven by the prior knowledge of the user about what trajectory topology is expected in the data. For the majority of use cases, the user will know very little about the expected trajectory, except perhaps whether the data is expected to contain multiple trajectories, cycles or a complex tree structure. In each of these use cases, a different set of methods performed optimally, with Monocle DDRTree performing best when the data contained a complex tree, and Slingshot performing equally well on less complex trajectories . No methods dealing with multiple trajectories or cycles were included in the current version of the evaluation, although AGA[61] for disconnected trajectories, and Waddington-OT[60] and AGA[61] for regular graphs are currently the only methods in literature able to handle these types of trajectories. In the case where the user would know the exact expected topology, our evaluation suggests the use of reCAT for cycles, SCORPIUS for linear trajectories, and Slingshot for bifurcating trajectories, although Slingshot could return other topologies if it would fit the data more accurately. The most difficult use case is when the topology is known but more complex than a bifurcating or cyclic trajectory. Here, to our knowledge, only ElPiGraph.R github.com/Albluca/ElPiGraph.R, which is not yet published, can be used.

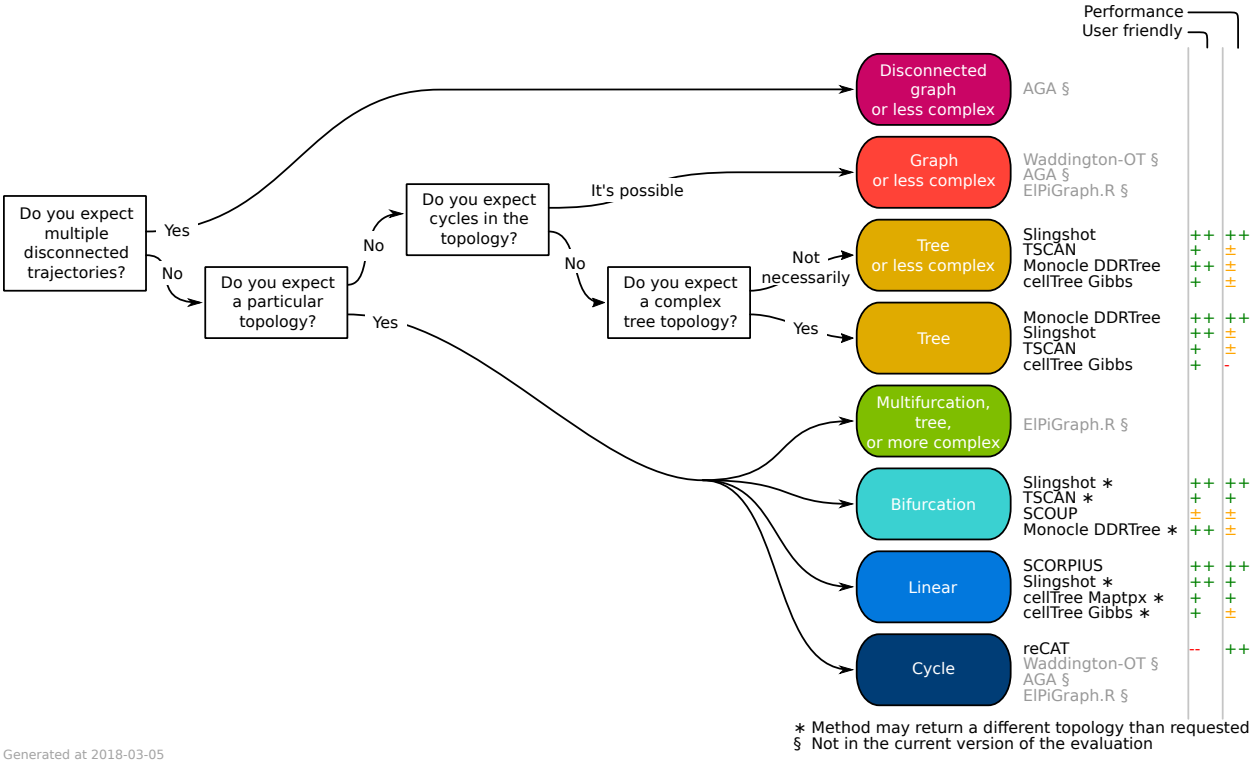


Figure 5 Practical guidelines for method users. As the performance of a method most heavily depended on the topology of the trajectory, the choice of TI method will be primarily influenced by the user's existing knowledge about the expected topology in the data. We therefore devised a set of practical guidelines, which combines the method's performance, user friendliness and the number of assumptions a user is willing to make about the topology of the trajectory. Methods to the right are ranked according to overall performance. Further to the right are shown the user friendly scores (++: ≥ 0.9, +: ≥ 0.8, ± ≥ 0.65, - ≥ 0.5) and overall performance (++: top method, +: difference between top method's performance ≥ -0.05, ±: ≥-0.2, -: ≥ -0.5).

When choosing a method, it is important to take two further points into account. First, it is important that a trajectory and the downstream results and/or hypotheses originating from it are confirmed by multiple TI methods. This to make sure the model is not biased towards the particular model underlying a TI method, for example its preferred trajectory type. Second, even if the expected topology is known, it can be beneficial to also try out methods which make the less assumptions about the trajectory topology. When the expected topology is confirmed using such a method, it provides extra evidence to the user's knowledge of the data. When a more complex topology is produced, this could indicate the presence of a more complex trajectory in the data than was expected by the user.

Discussion

Trajectory inference is unique among most other categories of single-cell analysis methods, such as clustering, normalisation and differential expression, because it models the data in a way that was almost impossible using bulk data. Indeed, for no other single-cell analysis types have so many tools been developed, according to several repositories such as omictools.org[73], the "awesome single cell software" list[74] and scRNA-tools.org[75]. It is therefore critical that these methods, now reaching 59, are evaluated to guide users in their choice. In this preprint, we present an initial version of our evaluation of these methods, focusing on the quality control and the accuracy of their model using the default parameters. When comparing the maximum overall score over time, it is encouraging to see multiple incremental improvements over the state-of-the-art (**Supplementary Figure 15**). We believe that the benchmarking presented in this study will pave the way to the next series improvements in the field of trajectory inference.

In this study, we presented our first version of the evaluation of TI methods. We are convinced that the results we provided will be useful for both method users and tool developers, as we provide clear practical guidelines for users depending on their current knowledge of the trajectory, as well as an objective benchmark on which new methods can be tested. Nonetheless, our evaluation can be expanded on several points, all of which we will try to tackle in the near future:

- Inclusion of methods published during of after june 2017
- A parameter tuning for each method, both across all trajectory types, as well as on specific trajectory types.
- A test of robustness on noise, parameter changes and dataset size
- An evaluation of the stability of a method's results when running the same parameter setting on the same dataset multiple times
- Evaluate the methods when given optional priors, and compare with performance without priors
- Assess the effect of noisy or incorrect prior information
- Test methods on datasets with no clear trajectory present
- Include feedback on the quality control scoring scheme, and update the scoring when method get updated
- Test the scalability of each method, both in the number of cells and the number of features
- Include more real datasets with complex trajectory types, as they become available
- Evaluate methods on other single-cell -omics datasets, such as proteomics and epigenomics data
- Provide functionality for visually interpreting and comparing predicted trajectories

Our evaluation indicated a large heterogeneity in the performance of the current TI methods, with Slingshot, TSCAN, and Monocle DDRTree, towering above all other methods. Nonetheless, we found that methods which did not perform well across all settings could still be useful in certain specific use cases. Indeed, on data containing more than one bifurcations, Monocle DDRTree clearly performed better than other methods. We found that this particular result was mainly caused by the fact that the default parameters of Monocle DDRTree preferably led to the detection of tree topologies, while those of Slingshot preferably found linear and bifurcation trajectories.

We managed to wrap the output of all methods into one common format. This not only allowed us to compare different methods with a gold standard, but could also be useful for TI users, as it allows the user to test multiple methods on the same data and compare the results without manual conversion of input and output. Furthermore, it makes it possible to directly compare the output of different methods, which opens up possibilities for new comparative visualisation techniques or ensemble methods. However, we acknowledge that our model has some limitations. Currently, It cannot take into account uncertainty of a cell's position, which can both occur on the cellular ordering (e.g. when the position of a cell is uncertain within a branch) or on the trajectory topology (e.g. when the connections between branches are uncertain). Some current methods already model this uncertainty in some way, mainly on the cellular ordering[30], and in the future we will adapt our output model to also allow this uncertainty.

The use of synthetic data to evaluate TI methods offers the advantage of having an exact gold standard to which the methods' results can be compared. However, the use of synthetic data can also be questionable, because the model which generates the data does not necessarily reflect the intrinsic characteristics of true biological systems. This could bias an evaluation towards methods for which the underlying model best fits the model used for generating the data. Therefore, it is essential that results of an evaluation on synthetic data are confirmed using real data. In our study, the overall performance of methods was very similar between real and synthetic datasets, confirming the biological relevance of the synthetic data. Given this, we believe that our synthetic data can now be used to effectively prototype new TI methods for more complex trajectories such as disconnected graphs, for which the availability of real datasets is poor. Furthermore, it is expected that in the future, methods will be able to model even more complex cellular behaviors, such as multiple dynamic processes happening at parallel in a single cell, the integration of datasets from different patients or trajectories in a spatial context[1,4]. Synthetic data generated with our workflow could therefore be used to spearhead the development of these methods, given that currently only a limited number of real datasets are available for which the methods could be useful. Furthermore, we believe that our data generation workflow could also be used to evaluate other types of single-cell modelling techniques, such as single-cell network inference, clustering and normalisation. We sincerely hope that such efforts will lead to a more rapid development of accurate methods, and will in the near future provide a package which can be used to simulate synthetic data for a wide variety of single-cell modelling problems.

Methods

Trajectory inference methods

Method selection

We gathered a list of 59 trajectory inference methods (**Table 1**), by searching in literature for "trajectory inference" and "pseudotemporal ordering", and based on two existing lists found online[74,76]. A continuously updated list can also be found online). We welcome any contributions by creating an issue at github.com/dynverse/dynverse/issues.

Methods were excluded from the evaluation based on several criteria: (a) Not free, (b) Unavailable, (c) Superseded by another method, (d) Requires data types other than expression, (e) No programming interface, (f) Unresolved errors during wrapping, (g) Too slow (requires more than one hour on a 100x100 dataset), (h) Doesn't return an ordering, (i) Requires additional user input during the algorithm (not prior information), (j) Published later than 2017-05-01 to be included in the current version of the evaluation, (k) This method is not published in preprint or a peer-reviewed journal. This resulted in the inclusion of 29 methods in the evaluation (**Table 1**).

Method input

As input, we provided for each method either the raw count data (after cell and gene filtering) or normalised expression values, based on the description in the methods documentation or from the study describing the method. Furthermore, when required, we also provided a maximum of 7 types of prior information. This prior information was extracted from the gold/silver standards as follows:

- **Start cells** The identity of one or more start cells. For both real and synthetic data, a cell was chosen which was the closest (in geodesic distance) from each milestone with only outgoing edges. For ties, one random cell was chosen. For cyclic datasets, a random cell was chosen.
- **End cells** The identity of one or more end cells. Similar as the start cells, but now for every state with only ingoing edges.
- **# end states** Number of terminal states. Number of milestones with only ingoing edges.
- **Grouping** For each cell a label to which state/cluster/branch it belongs. For real data, the states from the gold/silver standard. For synthetic data, each milestone was seen as one group, and cells were assigned to their closest milestone.
- **# branches** Number of branches/intermediate states. For real data, the number of states in the gold/silver standard. For synthetic data, the number of milestones.
- **Time course** For each cell a time point from which it was sampled. If available, directly extracted from the gold standard. For synthetic data: four timepoints were chosen, at which the cells were "sampled" to provide a time course information reflecting the one provided in real experiments.

Common trajectory model

Due to the absence of a common format for trajectory models, most methods return a unique set of output formats with few overlap. We therefore created wrappers around each method (available at github.com/dynverse/dynmethods) and postprocessed its output into a common probabilistic trajectory model (**Supplementary Figure 16a**). This model consists of three parts. (i) The milestone network represents the overall network topology, and contains edges between different milestones and the length of the edge between them. (ii) The milestone percentages contain, for each cell, its position between milestones, and sums for each cell to one. (iii) The trajectory model also allows for regions of delayed commitment, where cells are allowed to be positioned between three or more connected milestones. Regions must be explicitly defined in the trajectory model. Per region, one milestone must be directly connected to all other milestones in the network.

Depending on the output of a method, we used different strategies to convert the output to our model (**Supplementary Figure 16b**). Special conversions are denoted by an *****, and will be explained in more detail below.

- **Type 1, direct:** GPfates, reCAT, SCIMITAR, SLICE* and Wishbone. These methods assign each cell to a branch together with a pseudotime value and a branch network. The branch network is used as the milestone network, the percentages of a cell are proportional with its branch pseudotime.
- **Type 2, linear pseudotime:** Embeddr, Ouija, Ouijaflow, Phenopath, Pseudogp, SCORPIUS, Topslam, Wanderlust and Waterfall. These methods return a pseudotime value for each cell. The milestone network will consist of a single edge between two milestones, where cells are positioned on the transition proportional to their pseudotime value.
- **Type 3, end state probability:** MFA* and SCOUP. These methods return a global pseudotime value and a probability for every end state. We use a single start state and add an edge to every end milestone each representing an end state. The global pseudotime then determines the distance from the begin milestone, the rest of the cell's position is calculated by distributing the residual percentage over the end states, proportionally to the end state probabilities.
- **Type 4, cluster assignment:** Mpath and SCUBA. These methods return a cluster assignment and a cluster network. The cluster network was used as milestone network, each cell received percentage 1 or 0 based on its cluster assignment.
- **Type 5, projection onto nearest branch:** DPT*, Slingshot, StemID and TSCAN. These methods returning a cluster assignment, cluster network and dimensionality reduction. We projected each cell on the closest point on the edges between its own cluster and neighbouring clusters within the dimensionality reduction. This projection allowed us to give each cell a pseudotime within the edge, which was then converted into our model as described above, using the cluster network as milestone network.
- **Type 6, cell graph:** cellTree Gibbs, cellTree Maptpx, cellTree VEM, Monocle DDRTree, Monocle ICA, Sincell* and SLICER. These methods return a network of connected cells, and determine which cells are part of the "backbone". One milestone is created for each cell that is part of the backbone and has a degree $\neq 2$. Cells are positioned on the closest segment that is part of the backbone.

Special conversions were necessary for certain methods:

- **DPT** We projected the cells onto the cluster network, consisting of a central milestone (this cluster contains the cells which were assigned to the "unknown" branch) and three terminal milestones, each corresponding to a tip point. This was then processed as described above.
- **Sincell** To constrain the number of milestones this method creates, we merged two cell clusters iteratively until the percentage of leaf nodes was below a certain cutoff, default at 25%. This was then processed as described above.
- **SLICE** As discussed in the vignette of SLICE, we ran principal curves one by one for every edge detected by SLICE. This local pseudotime was then processed as above.
- **MFA** We used the branch assignment as state probabilities, which together with the global pseudotime were processed as described above.

Real datasets

We gathered a list of real datasets by searching for "single-cell" at the Gene Expression Omnibus and selecting those datasets in which the cells are sampled from different stages in a dynamic process (**Supplementary Table 1** and **Supplementary Figure 4**). The scripts to download and process these datasets will be made available on our repository (github.com/dynverse/dynalysis). Whenever possible, we preferred to start from the raw counts data. These raw counts were all normalised and filtered using a common pipeline, discussed later. We determined a reference standard for every dataset using labelling provided by the author's, and classified the standards into gold and silver based on whether this labelling was determined by the expert using

the expression (silver standard) or using other external information (such as FACS or the origin of the sample, gold standard) (**Supplementary Table 1**).

Synthetic datasets

Our workflow to generate synthetic data is based on the well established workflow used in the evaluation of network inference methods[70,77] and consists of four main steps: network generation, simulation, gold standard extraction and simulation of the scRNA-seq experiment (**Supplementary Figure 5**). At every step, we took great care to mimic real cellular regulatory networks as best as possible, while keeping the model simple and easily extendable. For every synthetic dataset, we used a random real dataset as a reference dataset (from those described earlier), making sure the number of variable genes and cells were similar.

Network generation

One of the main processes involved in cellular dynamic processes is gene regulation, where regulatory cascades and feedback loops lead to progressive changes in expression and decision making. The exact way a cell choses a certain path during its differentiation is still an active research field, although certain models have already emerged and been tested in vivo. One driver of bifurcation seems to be mutual antagonism, where genes[78] strongly repress each other, forcing one of the two to become inactive[79]. Such mutual antagonism can be modelled and simulated[80,81]. Although such a two-gene model is simple and elegant, the reality is frequently more complex, with multiple genes (grouped into models) repressing each other[82].

To simulate certain trajectory topologies, we therefore designed module networks in which the cells follow a particular trajectory topology given certain parameters (**Supplementary Figure 17**). Two module networks generated linear trajectories (linear and linear long), two generated simple forks (bifurcating and converging), one generated a complex fork (trifurcating), one generated a rooted tree (consecutive bifurcating) and two generated simple undirected graphs (bifurcating loop and bifurcating convergence).

From these module networks we generated gene regulatory networks in two steps: the main regulatory network was first generated, and extra target genes from real regulatory networks were added. For each dataset, we used the same number of genes as were differentially expressed in the real datasets. 5% of the genes were assigned to be part of the main regulatory network, and were randomly distributed among all modules (with at least one gene per module). We sampled edges between these individual genes (according to the module network) using a uniform distribution between 1 and the number of possible targets in each module. To add additional target genes to the network, we assigned every regulator from the network to a real regulator in a real network (from regulatory circuits[83]), and extracted for every regulator a local network around it using personalized pagerank (with damping factor set to 0.1), as implemented in the `page_rank` function of the *igraph* package.

Simulation of gene regulatory systems using thermodynamic models

To simulate the gene regulatory network, we used a system of differential equations similar to those used in evaluations of gene regulatory network inference methods[77]. In this model, the changes in gene expression (x_i) and protein expression (y_i) are modeled using ordinary differential equations[70] (ODEs):

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

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

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

The activation function is inspired by a thermodynamic model of gene regulation, in which the promoter of a gene can be bound or unbound by a set of transcription factors, each representing a certain state of the promoter. Each state is linked

with a relative activation α_j, a number between 0 and 1 representing the activity of the promoter at this particular state. The production rate of the gene is calculated by combining the probabilities of the promoter being in each state with the relative activation:

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

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

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

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

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

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

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

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

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

$$\frac{dy_i}{dt} = r \times x_i - \Lambda \times y_i + \eta \times \sqrt{y_i} \times \Delta W_t$$

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

Similar to[70], we sample the different parameters from random distributions, given in **Supplementary Table 4**.

We converted each ODE to an SDE by adding a chemical Langevin equation, as described in[70]. These SDEs were simulated using the Euler–Maruyama approximation, with time-step $h = 0.01$ and noise strength $\eta = 8$. The total simulation time varied between 5 for linear and bifurcating datasets, 10 for consecutive bifurcating, trifurcating and converging datasets, 15 for bifurcating converging datasets and 30 for linear long, cycle and bifurcating loop datasets. The burn-in period was for each simulation 2. Each network was simulated 32 times.

Simulation of the single-cell RNA-seq experiment

For each dataset we sampled the same number of cells as were present in the reference real dataset, limited to the simulation steps after burn-in. Next, we used the Splatter package[84] to estimate the different characteristics of a real dataset, such as the distributions of average gene expression, library sizes and dropout probabilities. We used Splatter to simulate the expression levels $\lambda_{i,j}$ of housekeeping genes i (to match the number of genes in the reference dataset) in every cell j. These were combined with the expression levels of the genes simulated within a trajectory. Next, true counts were simulated using $Y'_{i,j} \sim \text{Poi}(\lambda_{i,j})$. Finally, we simulated dropouts by setting true counts to zero by sampling from a Bernoulli distribution using a dropout probability $\pi^D_{i,j} = \frac{1}{1+e^{-k(\ln(\lambda_{i,j})-x_0)}}$.

Gold standard extraction

Because each cellular simulation follows the trajectory at its own speed, knowing the exact position of a cell within the trajectory topology is not straightforward. Furthermore, the speed at which simulated cells make a decision between two or more alternative paths is highly variable. To estimate a cell's position during a simulation within the trajectory topology, we therefore used the known progression of the modules, given in **Supplementary Figure 17c**, as a backbone. We smoothed the expression in each simulation using a rolling mean with a window of 50 time steps, and then calculated the average module expression

along the simulation. We used dynamic time warping, implemented in the dtw R package[85,86], with an open end to align a simulation to all possible module progressions, and then picked the alignment which minimised the normalised distance between the simulation and the backbone. In case of cyclical trajectory topologies, the number of possible milestones a backbone could progress through was limited to 20.

Expression normalisation pipeline

We used a standard single-cell RNA-seq preprocessing pipeline which applies parts of the scran and scater Bioconductor packages[87]. The advantages of this pipeline is that it works both with and without spike-ins, and includes a harsh cell filtering which looks at abnormalities in library sizes, mitochondrial gene expression, and number of genes expressed using median absolute deviations (set to 3). We required that a gene was expressed in at least 5% of the cells, and that it should have an average expression higher than 0.05. Furthermore, we used the pipeline to select the most highly variable genes, using a false discovery rate of 5% and a biological component higher than 0.5. As a final filter, we removed both zero genes and cells until convergence.

Evaluation metrics

The importance of using multiple metrics to compare complex models has been stated repeatedly[88]. We defined three metrics for comparing the likeness of predicted trajectories to a gold standard (**Figure 1c**). Each metric assesses the performance of a different aspect of the trajectory (**Supplementary Figure 18**); (i) the correlation metric measures the similarity in pairwise cell-cell distances; (ii) the RF MSE metric assesses whether cell neighbourhoods are similar in both trajectories; and (iii) the edge flip scores assesses similarity in trajectory topologies.

Correlation between geodesic distances

The similarity in cell ordering between two trajectories is assessed by calculating the geodesic distances between each pair of cells for both trajectories. The definition of a geodesic distance between two cells part of the common trajectory model will be demonstrated using a toy example (**Supplementary Figure 19**).

The geodesic distance between two cells depends on whether they are (i) on the same transition, or (ii) in different transitions. In the first case, the distance is defined as the product of the difference in milestone percentages and the length of the transition they both reside on. For cells a and b in the example, $d(a, b)$ is equal to $1 \times (0.9 - 0.2) = 0.7$. In the latter case, the distances between the cells and all of their neighbouring milestones will be calculated. These distances in combination with the milestone network are used to calculate the shortest path distance between the two cells. For cells a and c in the example, $d(a, X) = 1 \times 0.9$ and $d(c, X) = 3 \times 0.2$, and therefore $d(a, c) = 1 \times 0.9 + 3 \times 0.2$.

According to the defined common trajectory model (**Figure 1b**), cells are also allowed to have a delayed commitment. In a region of delayed commitment, one milestone will be connected to all other milestones as per the milestone network. In this case, the distance between two cells both inside a region of delayed commitment is calculated as the manhattan distances between the milestone percentages weighted by the transition weights from the milestone network. For cells d and e in the example, $d(d, e)$ is equal to $0 \times (0.3 - 0.2) + 2 \times (0.7 - 0.2) + 3 \times (0.4 - 0.1)$, which is equal to 1.9. The distance between two cells where one is part of a region of delayed commitment is calculated similarly to the previous paragraph, by first calculating the between the cells and their neighbouring milestones first, then calculating the shortest path distances between the two.

Finally, calculating all pairwise distances between cells would scale poorly for trajectories with large numbers of cells. For this reason, a set of waypoint cells are defined *a priori*, and only the distances between the waypoint cells and all other cells is calculated, in order to calculate the correlation of geodesic distances of two trajectories. The waypoints are determined by viewing each milestone, transition and region of delayed commitment as a collection of cells, and sampling cells from the different collections weighted by the total number of cells within that collection. For calculating the correlation of geodesic distances between two trajectories, the distances between all cells and the union of both waypoint sets is computed. For the benchmark evaluation, the total number of waypoints sampled from a trajectory was one hundred.

Random forest prediction error

Although the correlation between geodesic distances directly assesses the position of the cells in the trajectory, a bad correlation does not directly imply that similar cells were not grouped together by the method, as illustrated in **Supplementary Figure 18b,c**. For example, certain methods will inevitably reach an incorrect ordering because they cannot handle the correct trajectory type, but these methods could still correctly place similar cells next to each other. We therefore also included a metric which looks at the local neighbourhood of each cell, and assesses whether this neighbourhood can accurately predict the position of this cell in the gold standard. We used a Random Forest regression, implemented in the *ranger* package[89] to separately predict milestone percentages of every cell in the gold standard, using the milestone percentages of these cells in the prediction as features. We then used the out-of-bag mean-squared error on these percentages to score each method's capability of predicting the correct neighbourhood of each cell.

Edge flip score

As a third independent score, we assessed the similarity between the milestone network topologies. We first simplified each network, by merging consecutive linear edges into one edge, and adding new milestones within self loops such that $A \rightarrow A$ would be converted $A \rightarrow B \rightarrow C \rightarrow D$, by adding an intermediate node to linear networks. Because we are interested in the overall similarity between two topologies irrespective of the direction of the edges, the network was made undirected. Next, we define the edge flip score as the minimal number of edges which should be added or removed to convert one network into the other, divided by the total number of edges in both networks. This problem is equivalent to the maximum common edge subgraph problem, a known NP-hard problem without a scalable solution[90]. We implemented a branch and bound approach for this problem, by first enumerating all possible edge additions and removals with the minimal number of edges (the edge difference between the two networks) and if none of the new networks was isomorphic, we tried out all solutions with additional two edge changes. To further limit the search space, we made sure the degree distributions between the two networks were similar, before assessing whether the two networks were isomorphic using the BLISS algorithm[91], as implemented in the R *igraph* package.

A comparison of edge flip scores between common trajectory topologies is illustrated in **Supplementary Figure 20**.

Aggregation of scores

Supplementary Figure 21 illustrates the full set of aggregations performed on the raw scores in order to arrive at the final ranking of methods. In order to make the scores produced by the different metrics comparable to one another, across datasets of varying difficulty, the raw scores are percentile rank transformed per metric per dataset to a $[0, 1]$ range. The arithmetic mean is calculated across replicates. In order to ensure a method only obtains a high score if it scores well on all three metrics, the harmonic mean across metrics is calculated. At this point, the aggregated scores of all methods across all datasets is combined. As there can be an overrepresentation of datasets of a certain trajectory type, first an arithmetic mean is calculated per trajectory type, followed by an overall arithmetic mean across all trajectory types, thus obtaining a ranking of the methods.

Benchmark

Method execution

Each execution of a method on a dataset was performed in a separate task as part of a gridengine job. Each task was allocated one CPU core of an Intel(R) Xeon(R) CPU E5-2665 0 @ 2.40GHz, and was provided a maximum of 32GB in memory and 6 hours of wall time. One R session was started for each task, with the environment variable `R_MAX_NUM_DLLS` set to 500. The `base::set.seed` was overridden in order to prevent stochastic TI methods from pretending to be deterministic and/or robust. Timings of methods were measured for different steps along the executions, including preprocessing, postprocessing, each of the different metrics, and the method itself.

Effect of prior information

To assess the effect of prior information on the performance of a method, we compared the performance of methods which do or do not require the prior information using a two-tailed Mann–Whitney U test. P-values were controlled for multiple testing using Benjamini-Hochberg correction.

Effect of algorithm components

To assess the effect of algorithm components on the performance of a method, we used (1) a two-tailed Mann–Whitney U test, as implemented in the `wilcox.test` R function, and (2) increase in node purity importance scores using random forest classification, as implemented in the R randomForest package, predicting whether a method scored better than the median score. To evaluate whether a method increased or decreased performance, we used the estimate of the location parameter of the Mann–Whitney U test. We only investigated algorithm components which were part of at least 4 different methods in our evaluation study. P-values were controlled for multiple testing using Benjamini-Hochberg correction.

Method quality control

We created a transparent scoring scheme to check the quality of each method (**Supplementary Table 3**), based on several existing tool quality and programming guidelines in literature and online[92,71,93,94,72,95,96,97,98,99,100,101,102,103,104,105] . The goal of this quality control in the first place is to stimulate the improvement of current methods, and the development of user and developer friendly new methods. The quality control assessed 6 categories, each looking at several aspects, which are further divided into individual items. The availability category checks whether the method is easily available, whether the code and dependencies can be easily installed, and how the method can be used. The code quality assesses the quality of the code both from a user perspective (function naming, dummy proofing and availability of plotting functions) and a developer perspective (consistent style and code duplication). The code assurance category is frequently overlooked, and checks for code testing, continuous integration[99] and an active support system. The documentation category checks the quality of the documentation, both externally (tutorials and function documentation) and internally (inline documentation). The behaviour category assesses the ease by which the method can be run, by looking for unexpected output files and messages, prior information and how easy the trajectory model can be extracted from the output. Finally, we also assessed certain aspects of the study in which the method was proposed, such as publication in a peer-reviewed journal, the number of dataset in which the usefulness of the method was shown, and the scope of method evaluation in the paper.

Each aspect was further assigned to one or more applications, based on whether it influenced the user friendliness of the tool (such as code availability, good documentation or contains plotting functions), the developer friendliness of the tool (such as unit testing, inline documentation and a clear licensing), or indications that the tool will be broadly applicable on new datasets (such as being open-source, containing a good tutorial and support system, and being thoroughly evaluated in the study where it was published).

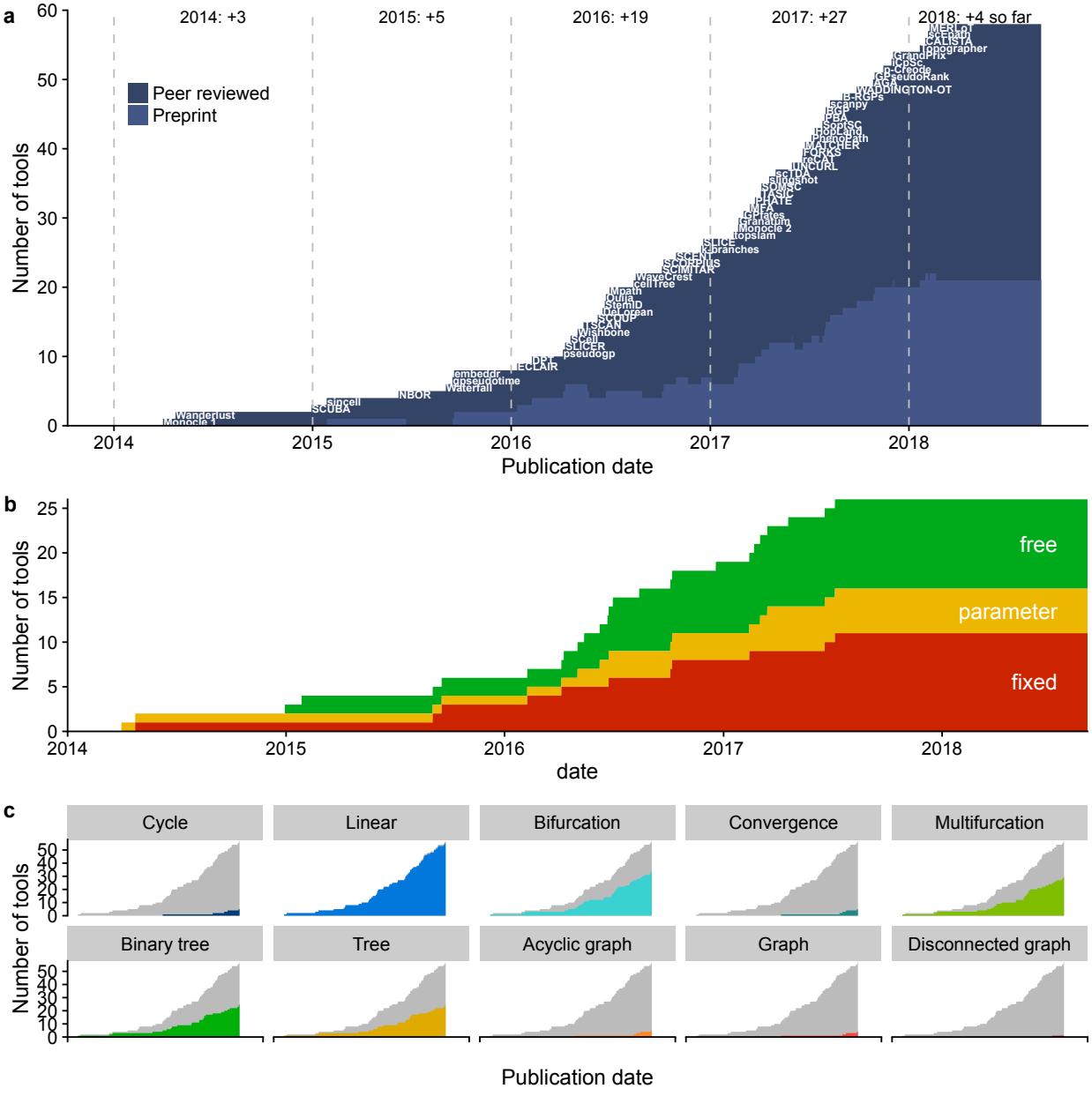
Each quality aspect received a weight depending on how frequently it was found in several papers and online sources which discuss tool quality (**Table 1**). This was to make sure that more important aspects, such as the open source availability of the method, outweighed other aspects, such as the availability of a graphical user interface. Within each aspect, we also assigned a weight to the individual questions being investigated (**Table 1**). For calculating the final score, we weighed each of the six categories equally.

Trajectory types

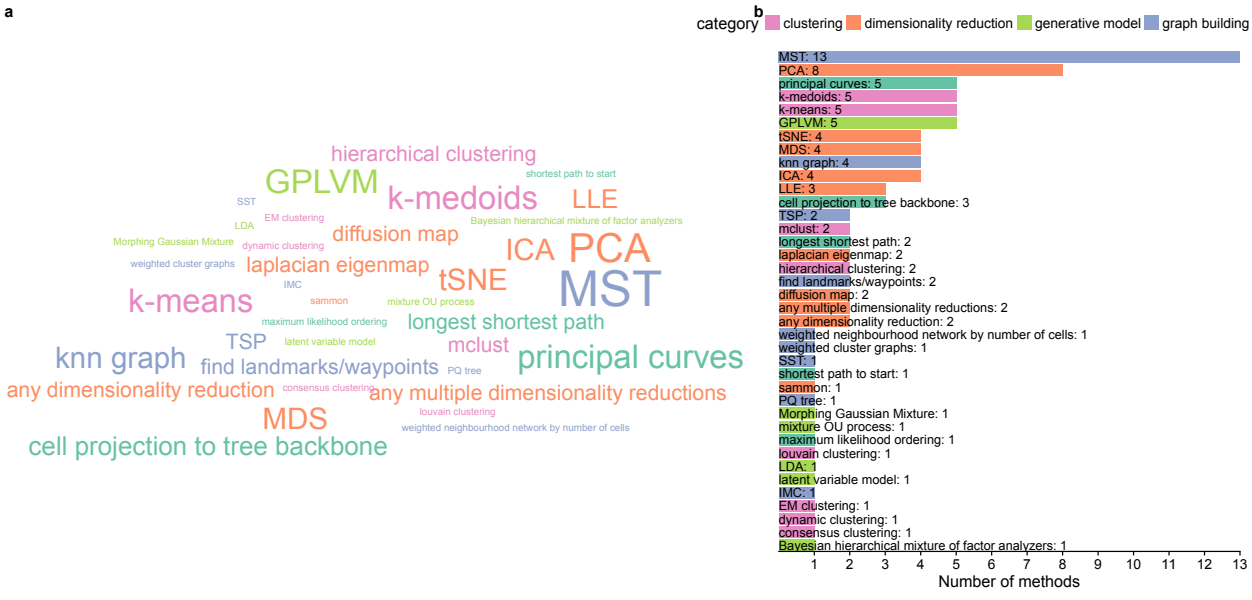
We classified all possible trajectory topologies into distinct trajectory types, based on topological criteria. These trajectory types start from the most general trajectory type, a disconnected directed graph, and move down (within a directed acyclic graph structure), progressively becoming more simple until the two basic types: linear and cyclical (**Supplementary Figure 22a**). For every directed trajectory type, a corresponding undirected trajectory type can also be defined (**Supplementary Figure 22b**). In most cases, a method which was able to detect a complex trajectory type, was also able to detect less complex trajectory types, with the only exception being DPT (limited to bifurcating trajectories).

Supplementary Material

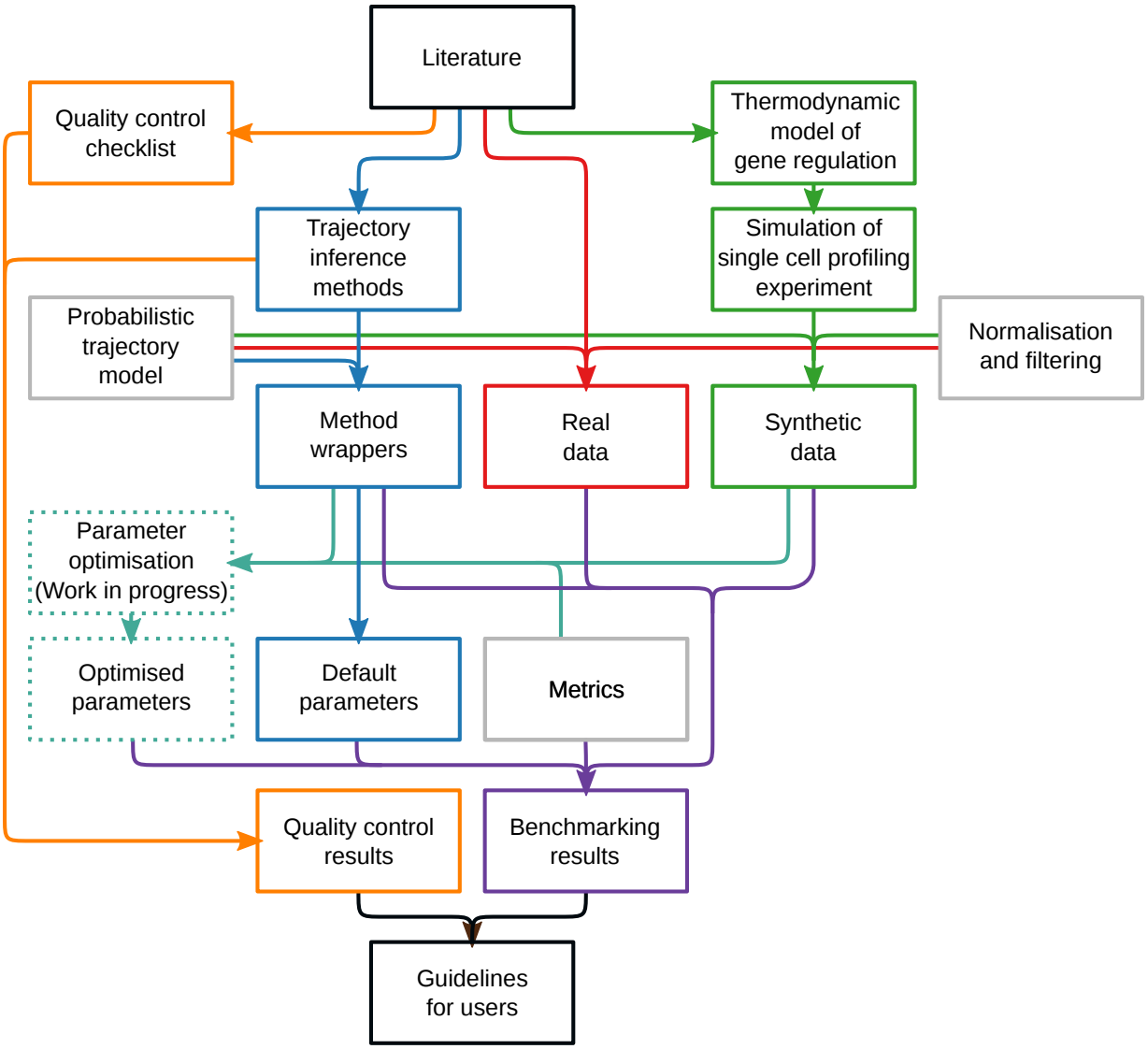
Supplementary Figures



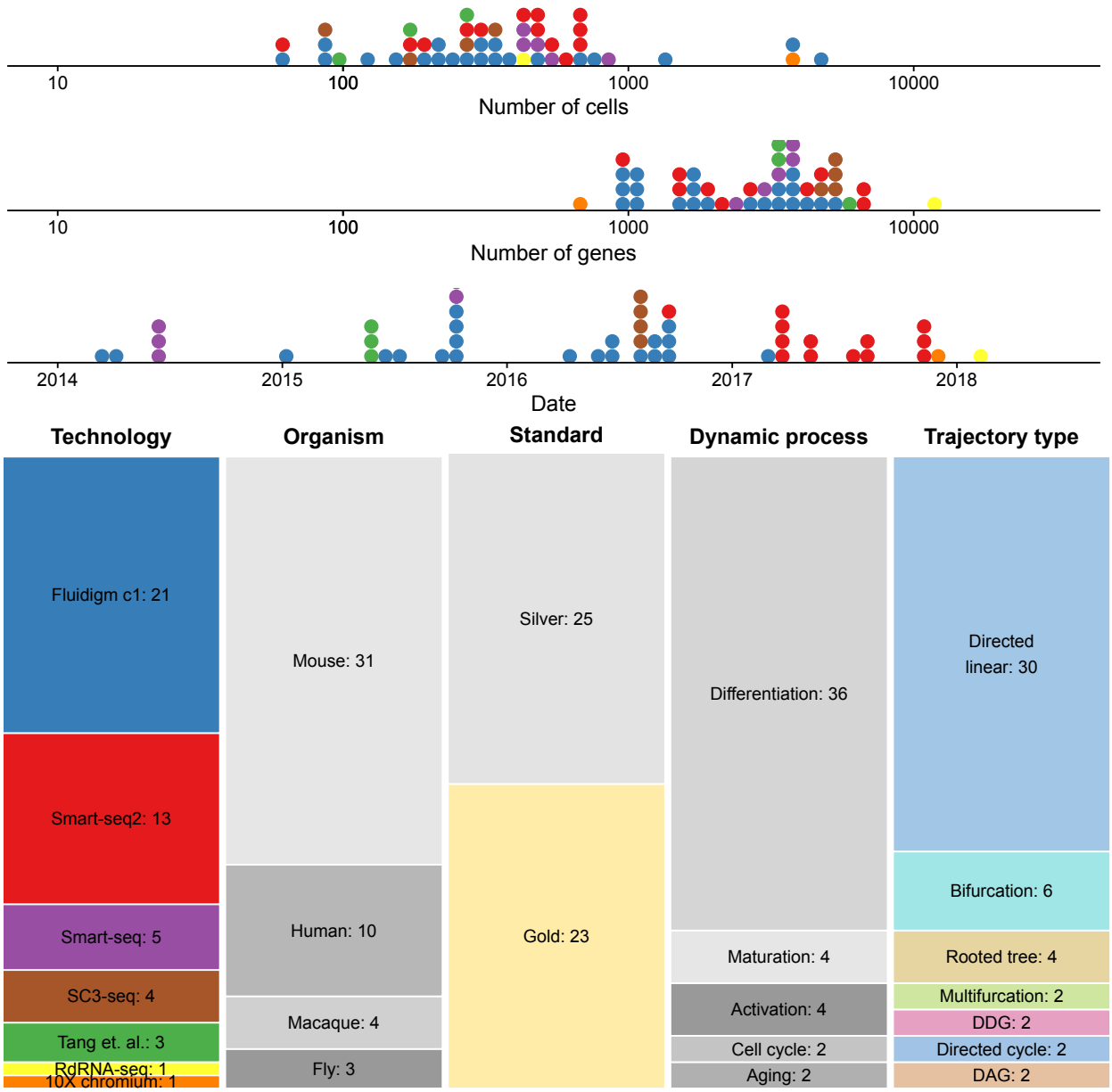
Supplementary Figure 1 New TI methods over time. a) Number of methods published or in preprint. b) Number of methods fixing the trajectory topology, either by design (fixed) or through user parameters (parameter). c) Number of methods which can handle a particular type of trajectory.



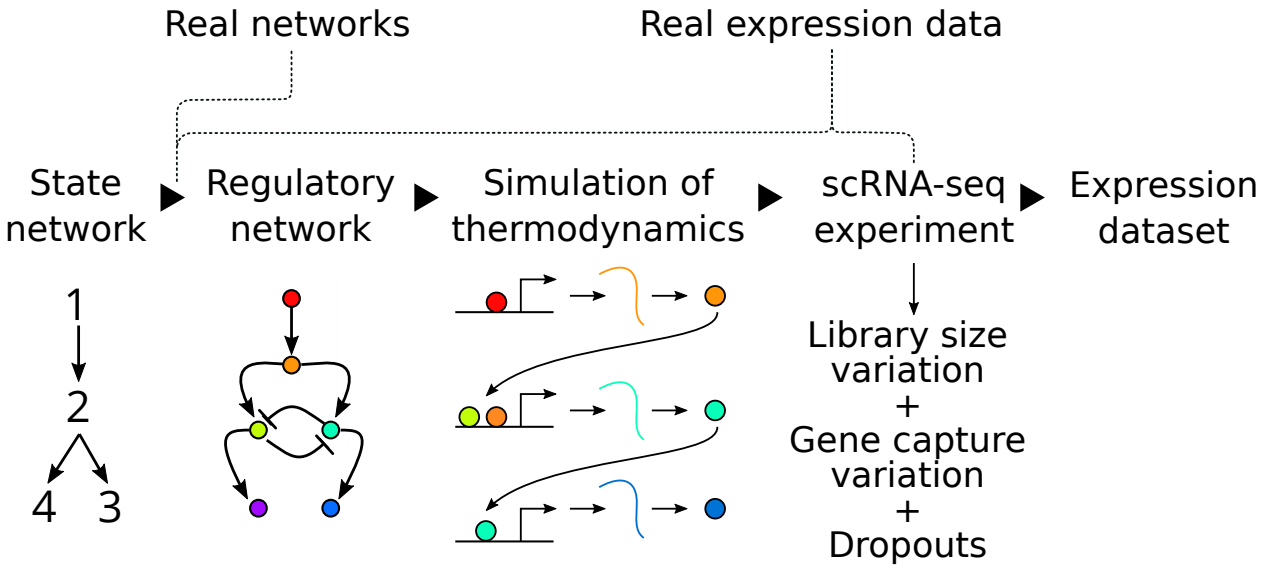
Supplementary Figure 2 Common algorithmic components shared by different TI methods. Most components can be categorised into 4 categories: clustering (pink), dimensionality reduction (orange), generative models (green) and graph building (blue). a) Wordcloud of the common components. b) Number of times a component was shared between methods. c) Number of methods containing a particular category.



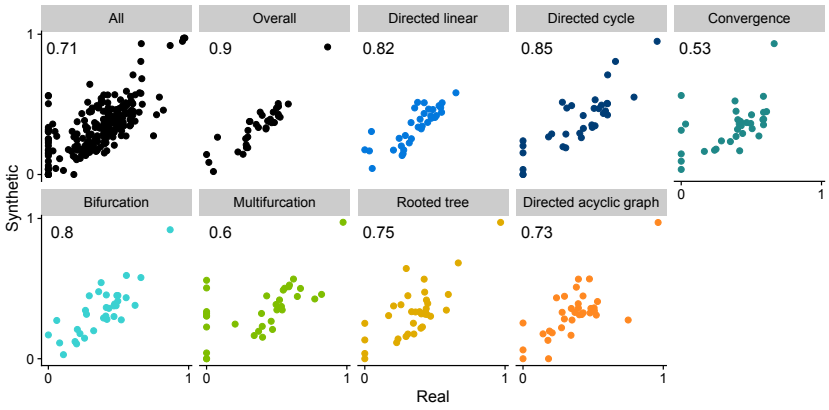
Supplementary Figure 3 Extended overview of our evaluation pipeline. From literature we extracted a quality control checklist, a list of trajectory inference methods, a set of real datasets containing a trajectory and real regulatory networks used to generate the synthetic data. We created a wrapper of each method, so that its output was transformed into a common probabilistic trajectory model, and used these to infer trajectories on all real and synthetic datasets using their default parameters. Using several similarity metrics, we compared the gold standard of the real and synthetic datasets with the inferred trajectories. We also evaluated the quality of each method using the quality control checklist, and used both the benchmark results and quality control results to produce a final set of guidelines for method's users.



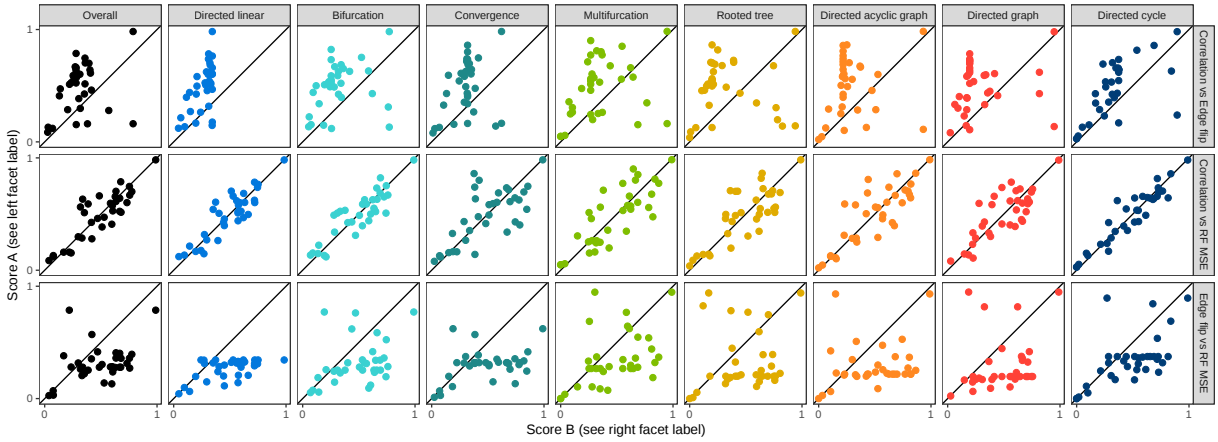
Supplementary Figure 4 Characteristics of the real datasets used for the evaluation. Top: distributions of number of cells and genes (after filtering) and the date these datasets were published. Bottom: Diversity of technologies which were used to generate the dataset, the organism, whether we extracted a silver or a gold standard from the data, the type of dynamic process and the trajectory type present in the data.



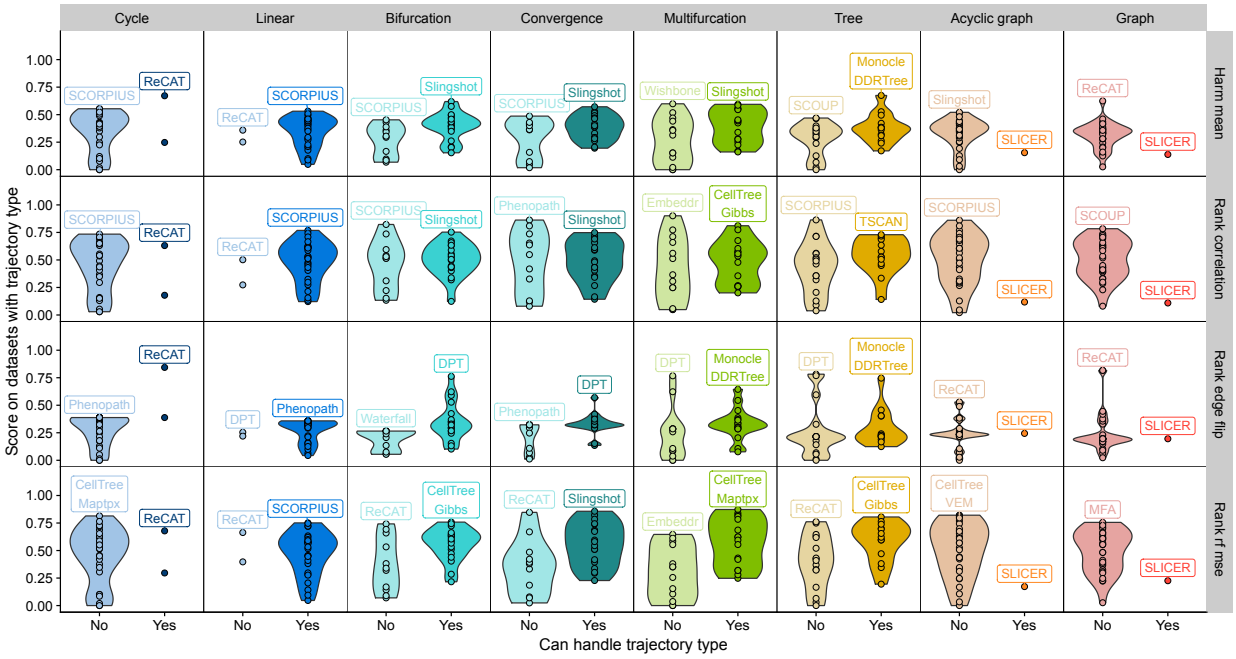
Supplementary Figure 5 Workflow to generate the synthetic data. Starting from a state network, we extracted a regulatory network using real networks which is expected to induce such a set of state transitions when simulated. We simulated this network using a detailed model of gene regulation which includes the thermodynamics of transcription factor binding. Finally, we simulated the single-cell RNA-seq (scRNA-seq) experiment itself, by matching the distribution of expression and dropouts to a real reference dataset.



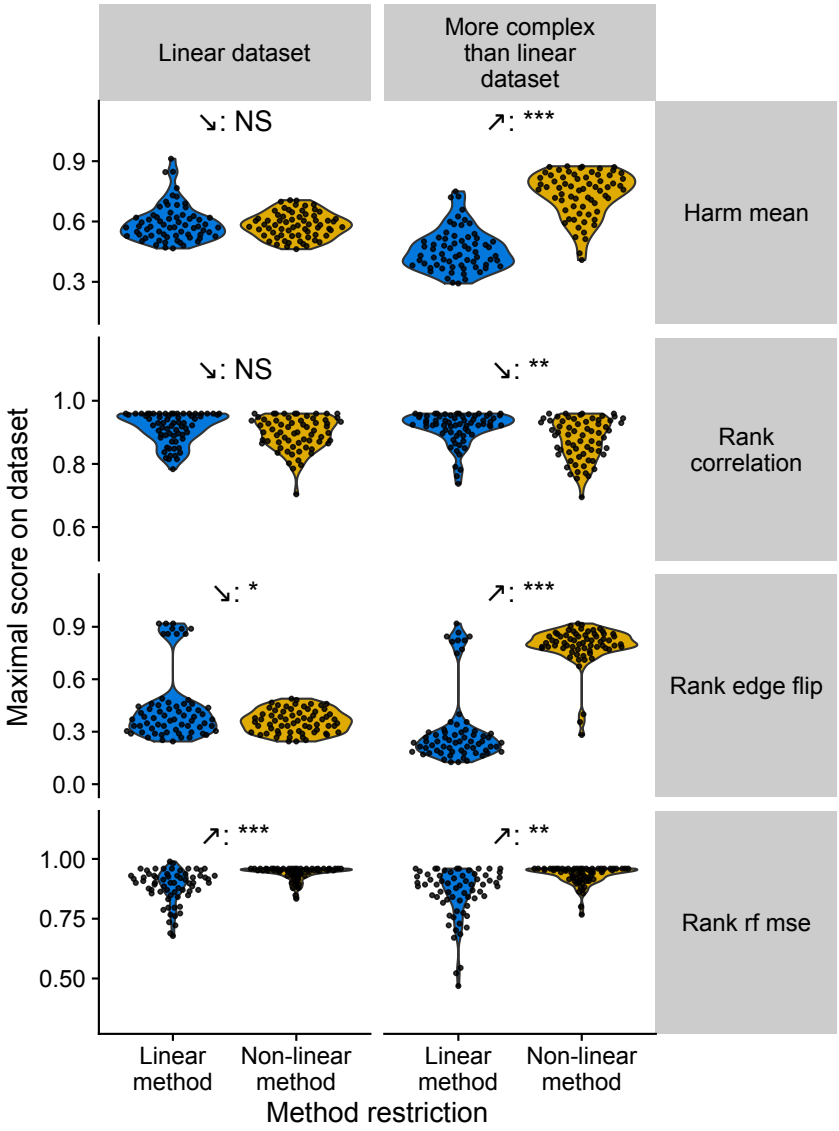
Supplementary Figure 6 Comparison between performance on real and synthetic datasets across trajectory types. Given in the top-left corner of each panel is the Pearson's correlation.



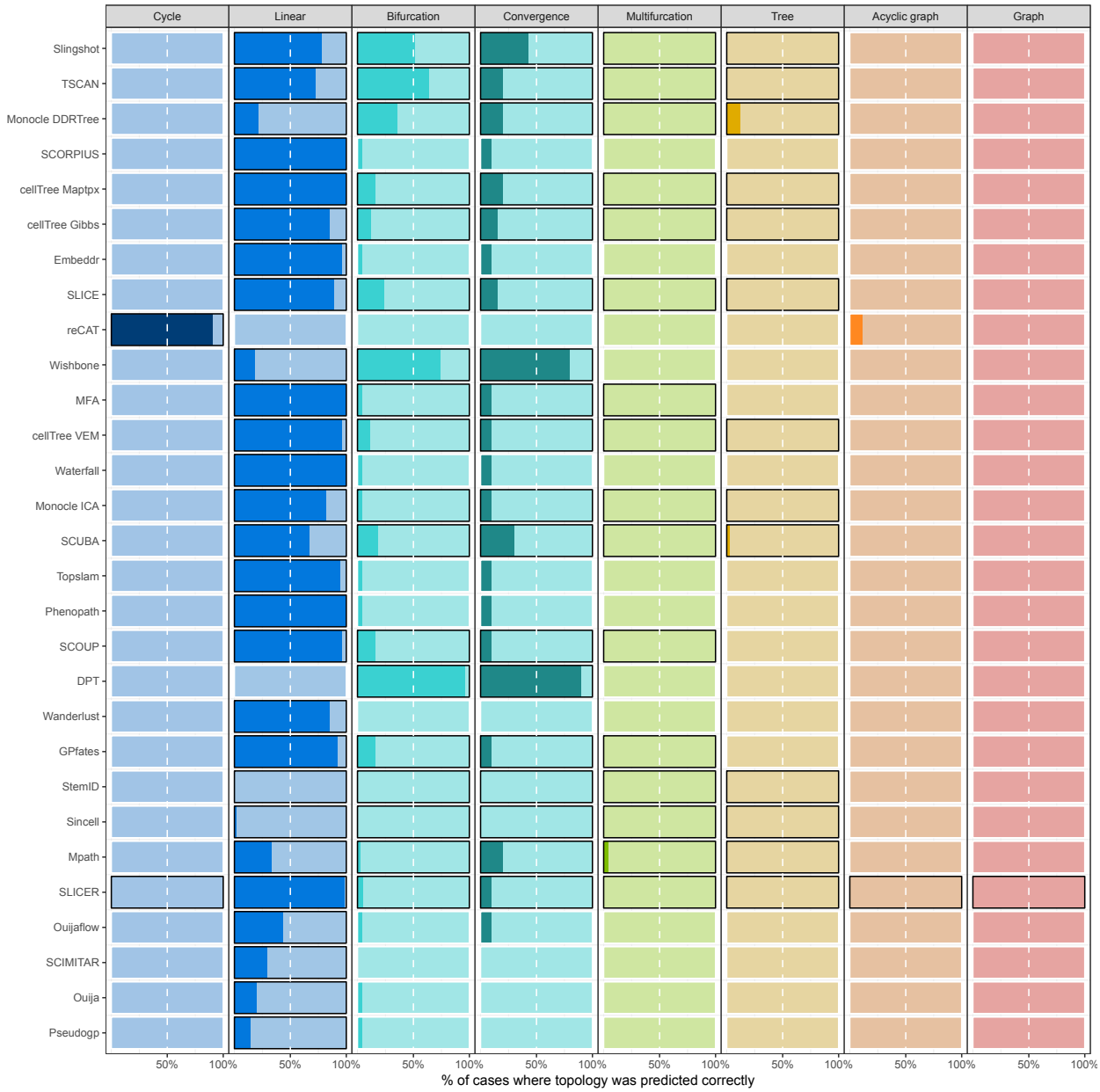
Supplementary Figure 7 Comparison between the different metrics across trajectory types. For each row, a different pairwise comparison is given between two metrics.



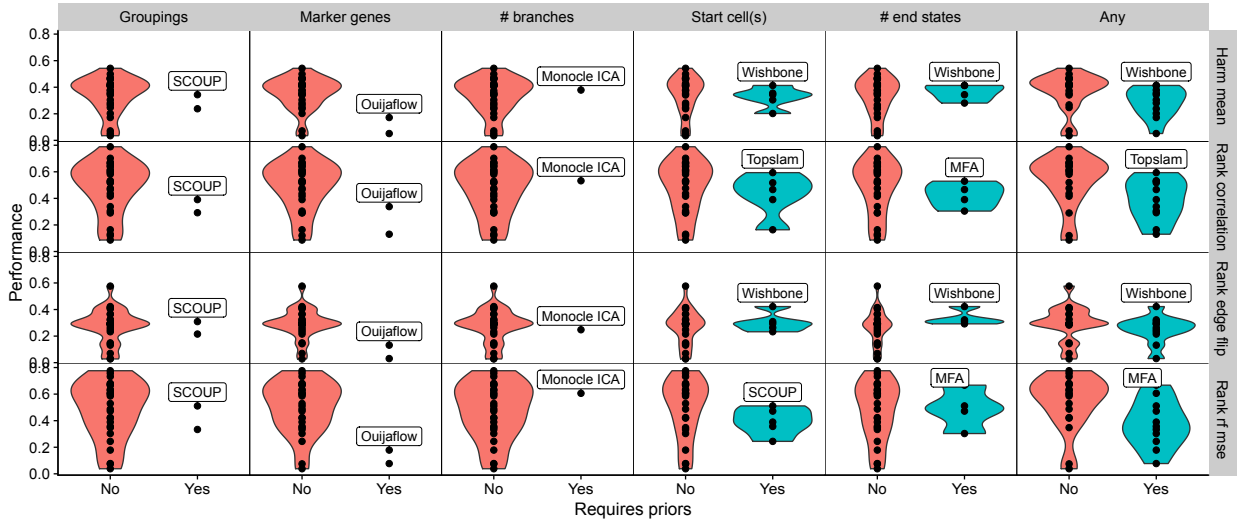
Supplementary Figure 8 Comparison between the performance of methods able to handle a trajectory type with those who don't. Shown is the performance on those datasets containing a particular trajectory type (top), according to several metrics (right).



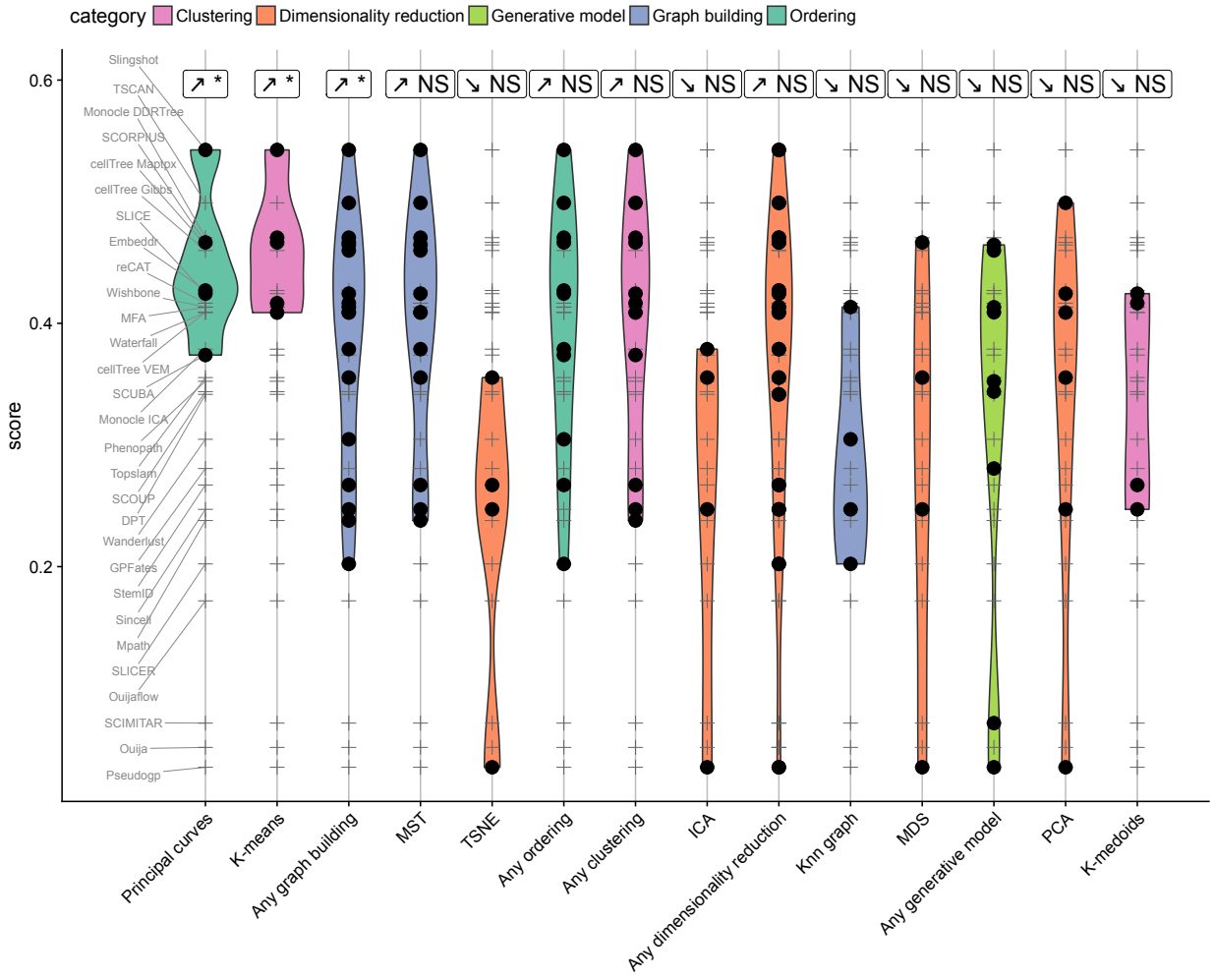
Supplementary Figure 9 Comparison of the performance between linear and non-linear methods. Methods were split in linear methods (those which can either only detect cyclic or linear trajectories) and non-linear methods. Their performance is shown on linear and non-linear datasets according to several performance metrics (right). NS: Not significant, *: p-value < 0.05, **: p-value < 0.001, ***: p-value < 0.0001



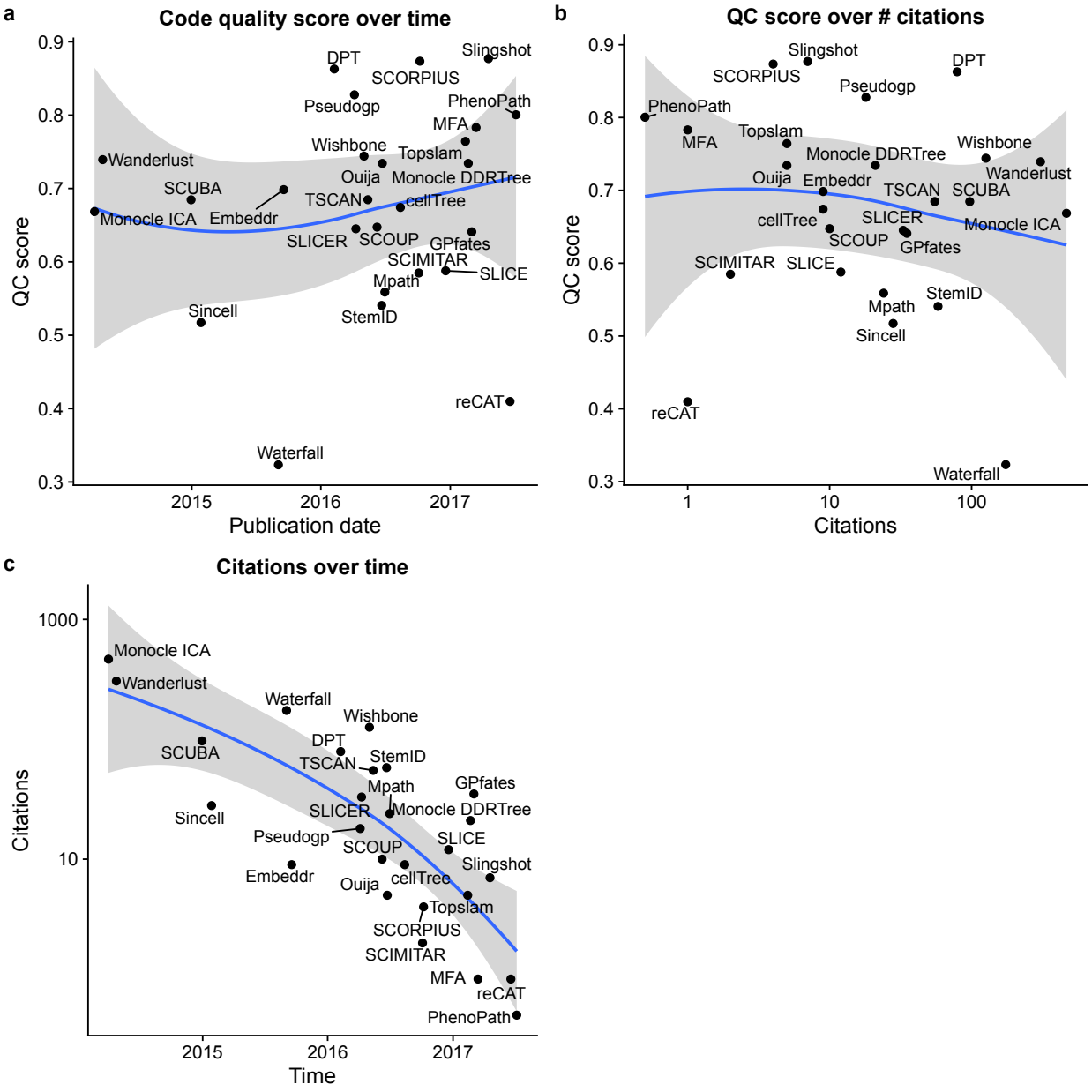
Supplementary Figure 10 Sensitivity of each TI method in predicting the correct trajectory topology. Shown are the % of datasets where a method is able to predict the exact correct trajectory topology, depending on the trajectory type present in the data. When a method is able to predict a particular trajectory type, the box is surrounded by a black border.



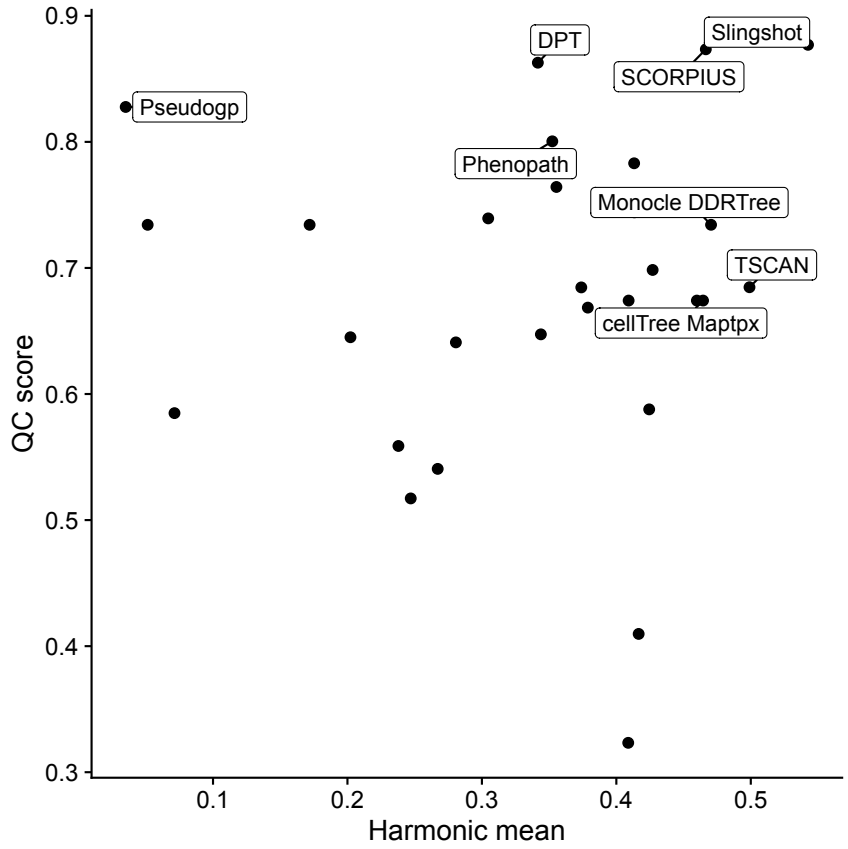
Supplementary Figure 11 Comparison of method performance depending on whether the method required a certain prior or not. The top performing method which requires a particular type of prior information is labelled.



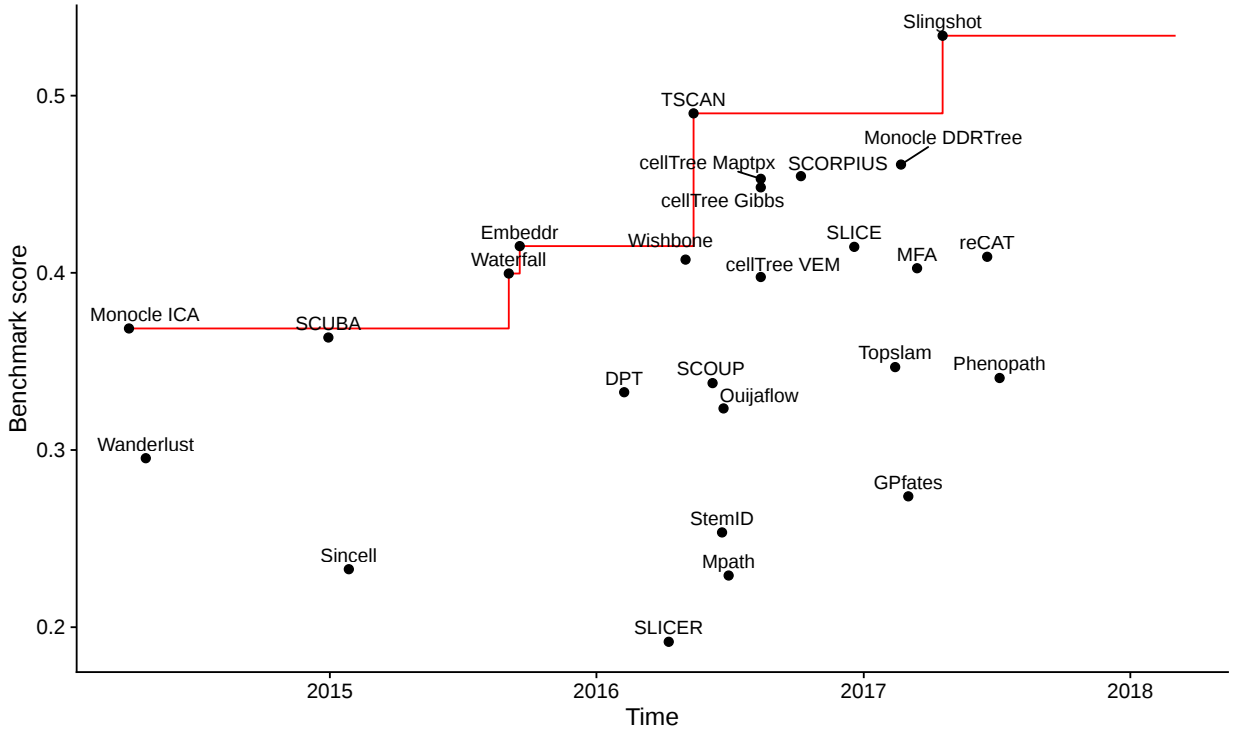
Supplementary Figure 12 Score of methods containing particular algorithmic components. Shown are the algorithmic components present in four or more methods, and whether the performance of methods containing a certain component was significantly higher than all the other methods. NS: Not significant. *: corrected p-value < 0.05



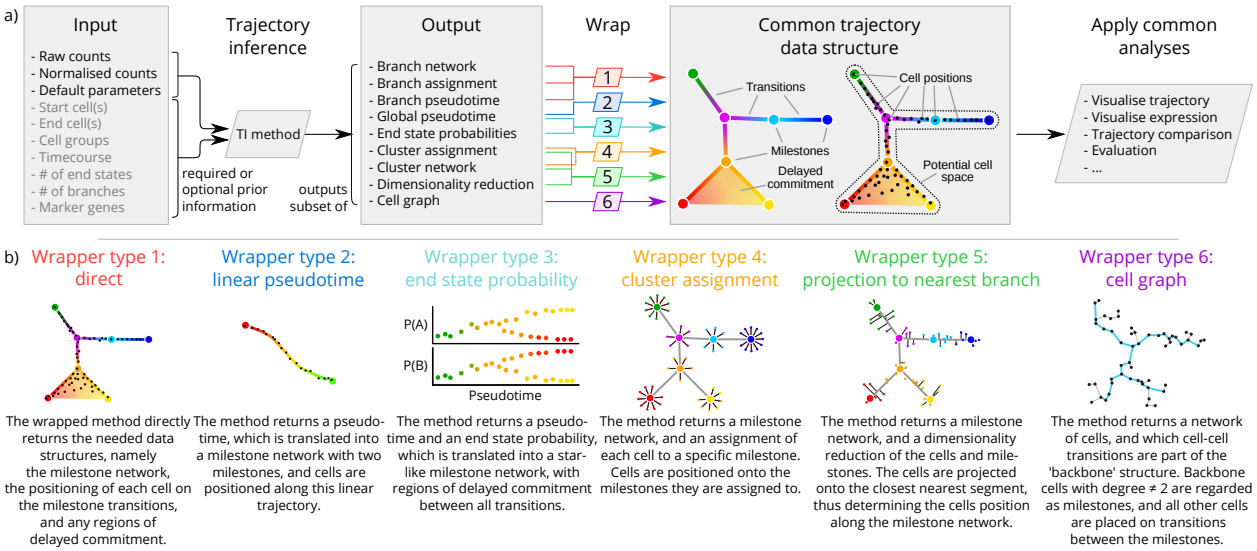
Supplementary Figure 13 Quality control scores and number of citations over time.



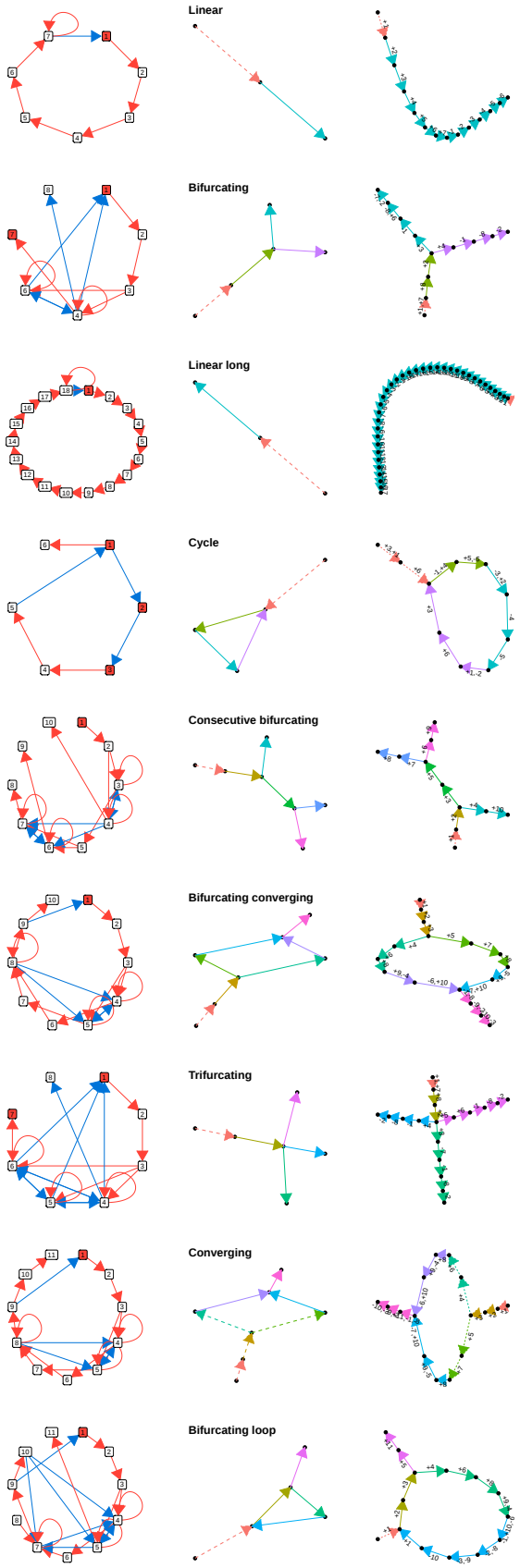
Supplementary Figure 14 Comparison between method quality control scores and overall method performance.



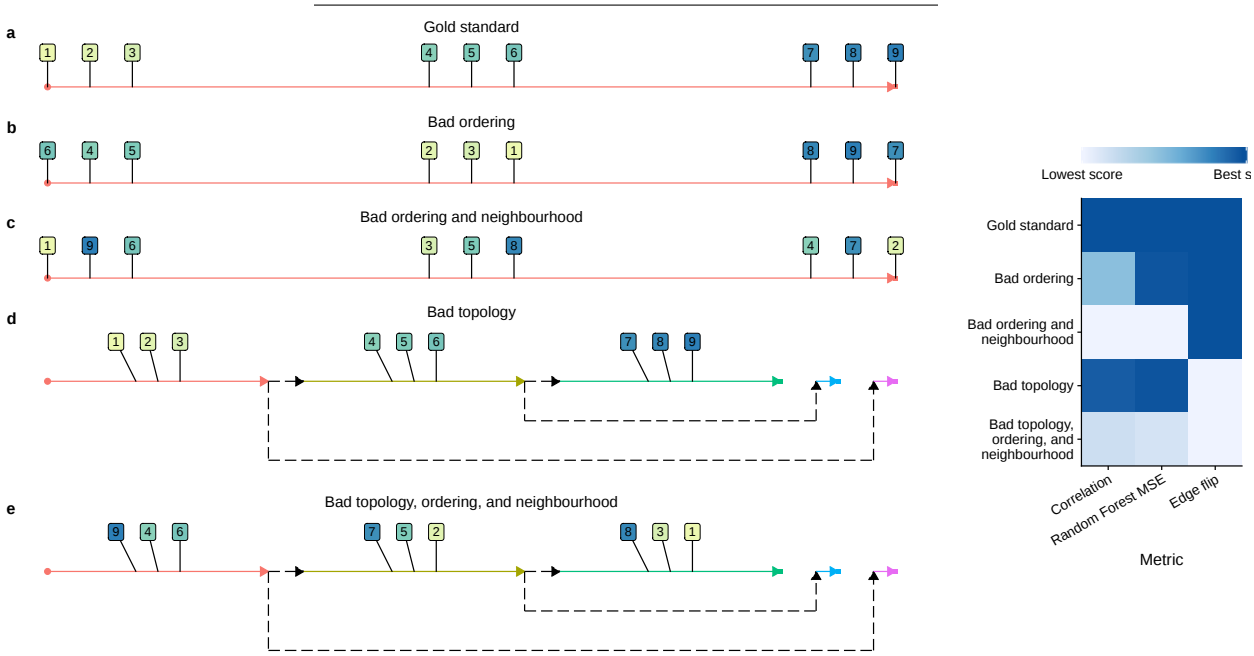
Supplementary Figure 15 Method performance compared to the time the method was published. The red line indicates the optimal performance up to a point in time.



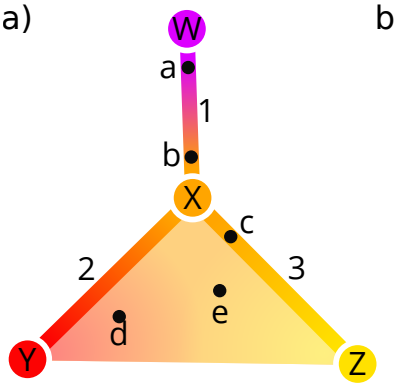
Supplementary Figure 16 A common interface for TI methods. a) The input and output of each TI method is standardised. As input, each TI method receives either raw or normalised counts, several parameters, and a selection of prior information. After its execution, a method uses one of the six wrapper functions to transform its output to the common trajectory model. This common model then allows to perform common analysis functions on trajectory models produced by any TI method. b) The specific transformations performed by each of the wrapper functions is explained in more detail.



Supplementary Figure 17 Module networks, milestone networks and the module dynamics for each of the different types of synthetic data



Supplementary Figure 18 Different scores assess different aspects of the correspondence between a prediction and gold standard trajectory.



b) Milestone network

from	to	length
W	X	1
X	Y	2
X	Z	3

Region of delayed commitment

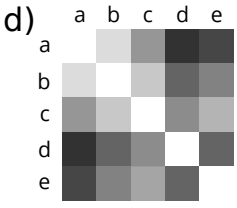
region	to	is_begin
XYZ	X	TRUE
XYZ	Y	FALSE
XYZ	Z	FALSE

Milestone percentages

cell	milestone	percentage
a	W	0.9
a	X	0.1
b	W	0.2
b	X	0.8
c	X	0.8
c	Z	0.2
d	X	0.2
d	Y	0.7
d	Z	0.1
e	X	0.3
e	Y	0.2
e	Z	0.4

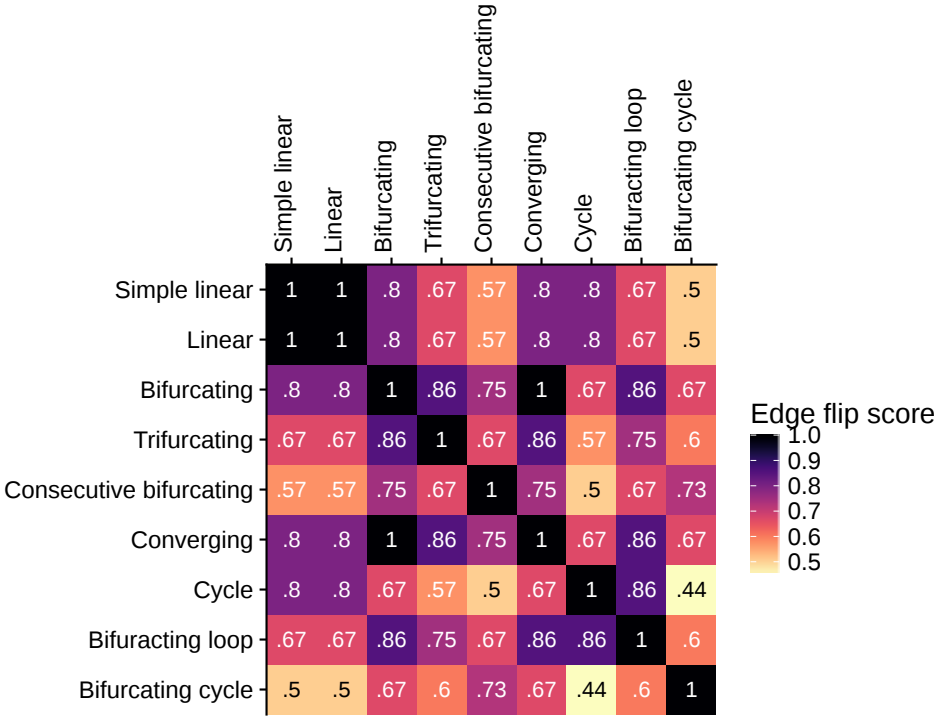
c)
d(a, b) = 1 × (0.9 - 0.2) = 0.7
d(a, c) = 1 × 0.9 + 3 × 0.2 = 1.5
d(b, c) = 1 × 0.2 + 3 × 0.2 = 0.8
d(a, d) = 1 × 0.9 + 2 × 0.7 + 3 × 0.1 = 2.6
d(b, d) = 1 × 0.2 + 2 × 0.7 + 3 × 0.1 = 1.9
d(c, d) = 2 × (0.7 - 0) + 3 × (0.2 - 0.1) = 1.7
d(a, e) = 1 × 0.9 + 2 × 0.2 + 3 × 0.4 = 2.5
d(b, e) = 1 × 0.2 + 2 × 0.2 + 3 × 0.4 = 1.8
d(c, e) = 2 × (0.2 - 0) + 3 × (0.4 - 0.2) = 1.0
d(d, e) = 2 × (0.7 - 0.2) + 3 × (0.4 - 0.1) = 1.9

d)



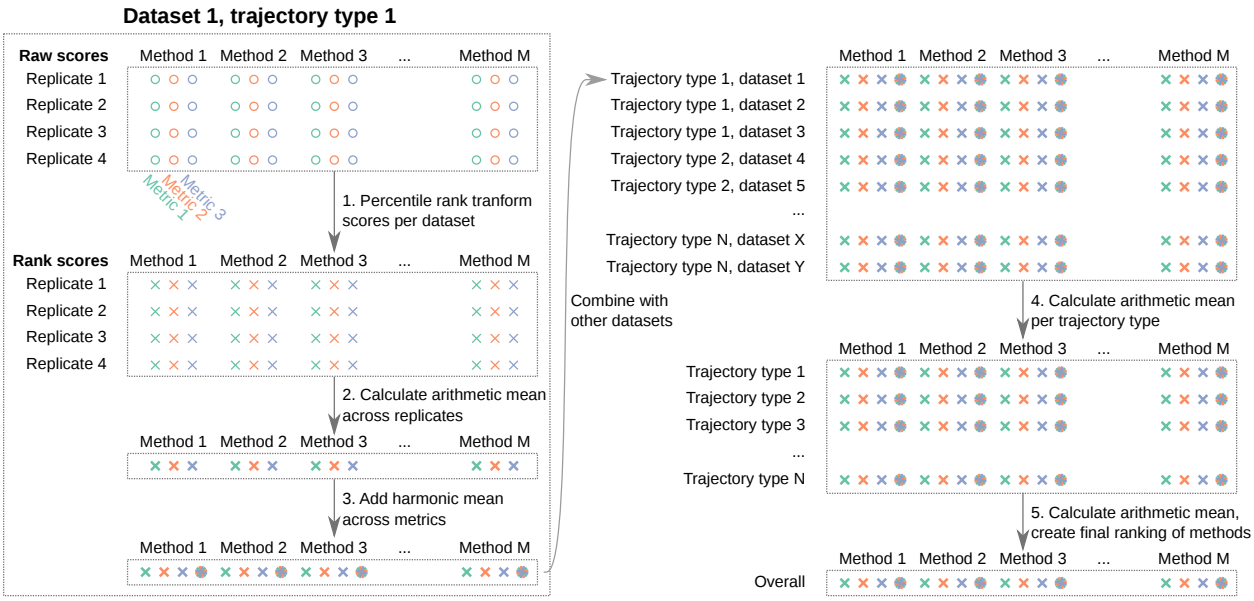
Supplementary Figure 19 The modified geodesic distances demonstrated on a toy example. a) A toy example containing four milestones (W to Z) and five cells (a to e). b) The corresponding milestone network, milestone percentages and regions

of delayed commitment, when the toy trajectory is converted to the common trajectory model. c) The calculations made for calculating the pairwise geodesic distances. d) A heatmap representation of the pairwise geodesic distances.

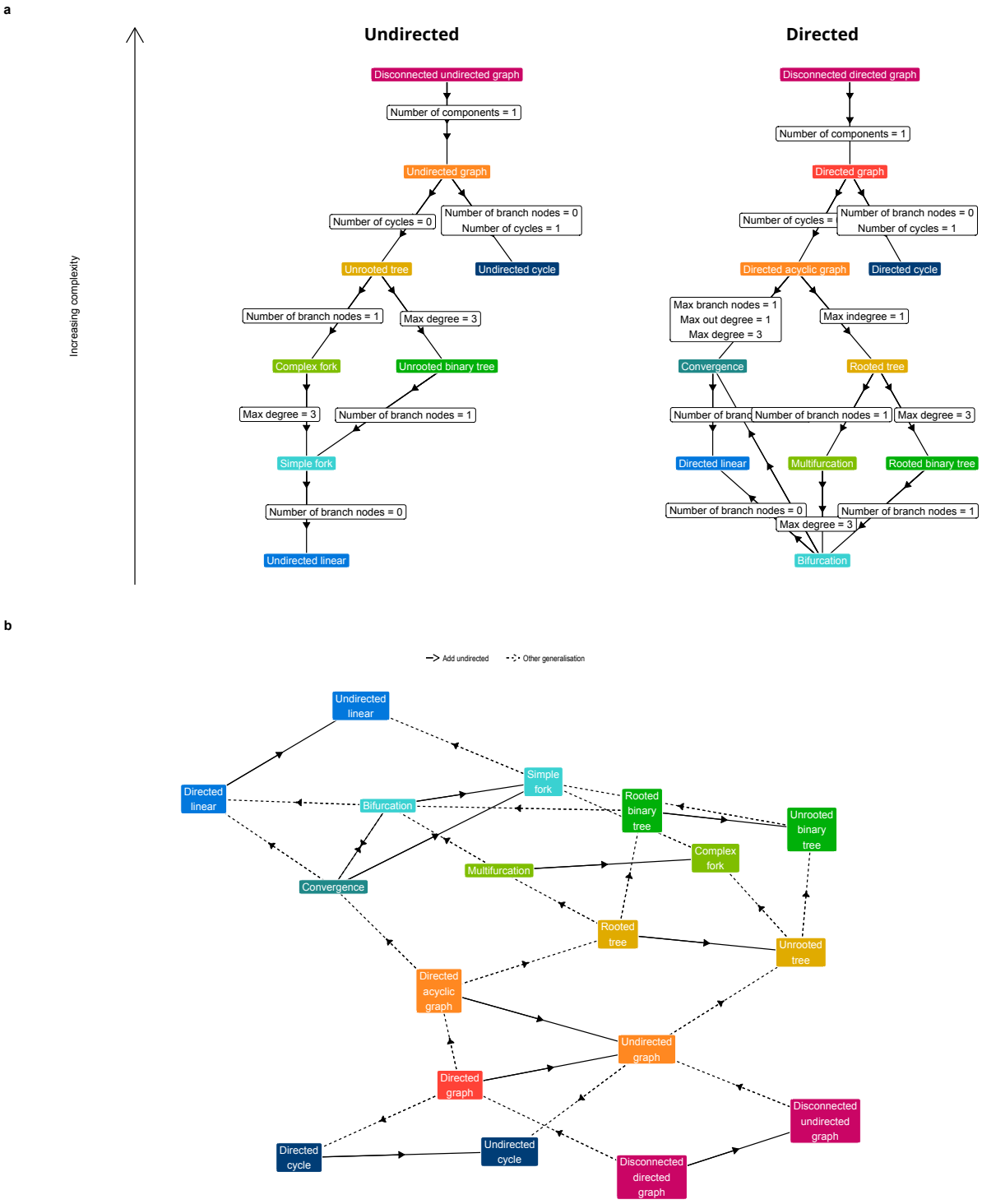


	Simple linear	Linear	Bifurcating	Trifurcating	Consecutive bifurcating	Converging	Cycle	Bifurcating loop	Bifurcating cycle
Simple linear	1	1	.8	.67	.57	.8	.8	.67	.5
Linear	1	1	.8	.67	.57	.8	.8	.67	.5
Bifurcating	.8	.8	1	.86	.75	1	.67	.86	.67
Trifurcating	.67	.67	.86	1	.67	.86	.57	.75	.6
Consecutive bifurcating	.57	.57	.75	.67	1	.75	.5	.67	.73
Converging	.8	.8	1	.86	.75	1	.67	.86	.67
Cycle	.8	.8	.67	.57	.5	.67	1	.86	.44
Bifuracting loop	.67	.67	.86	.75	.67	.86	.86	1	.6
Bifurcating cycle	.5	.5	.67	.6	.73	.67	.44	.6	1

Supplementary Figure 20 Edge flip score values for common trajectory topologies, defined in **Supplementary Figure 17**.



Supplementary Figure 21 Aggregation methodology of metric scores.



Supplementary Figure 22 Different trajectory types can be classified in a directed acyclic graph structure, in which certain types are a generalisation of other trajectory types. a) The graph structure for undirected and directed trajectory types. Shown are the topological properties which change when going from a particular trajectory type to a less complex type. b) Combined structure for directed and undirected trajectory types.

Supplementary Tables

Supplementary Table 1 Real datasets used in this study.

Source(s)	Organism	Technology	Trajectory type	Standard determination	# cells	# genes
GSE52529	mouse	fluidigm c1	Directed linear	timeseries	291	3582
GSE52583	mouse	fluidigm c1	Multifurcation	clustering: hierarchical	65	1095
		smart-seq	Directed linear	timeseries	541	3630
GSE48968	mouse	smart-seq	Directed linear	timeseries	408	3387
		smart-seq	Directed linear	timeseries	435	3604
E-MTAB-2805	mouse	fluidigm c1	Directed cycle	FACS	264	5310
		Tang et. al.	Bifurcation	timeseries, sample origin	272	5708
GSE63818	human	Tang et. al.	Directed linear	timeseries, sample origin	166	3457
		Tang et. al.	Directed linear	timeseries, sample origin	101	3502
GSE64016	human	fluidigm c1	Directed cycle	clustering: oscope	222	3766
GSE60781	mouse	fluidigm c1	Directed linear	FACS	238	1845
E-MTAB-3929	human	fluidigm c1	Directed linear	timeseries	1299	4135
GSE59114	mouse	smart-seq	Directed linear	FACS	873	2863
		smart-seq	Directed linear	FACS	493	2406
		fluidigm c1	Directed acyclic graph	clustering	158	1737
GSE71982	mouse	fluidigm c1	Directed linear	clustering	117	1673
		fluidigm c1	Directed linear	clustering	85	1584
		fluidigm c1	Directed linear	clustering	85	1648
GSE74596	mouse	fluidigm c1	Bifurcation	FACS	197	3982
GSE67310	mouse	fluidigm c1	Bifurcation	timeseries, clustering: PCA + hierarchical	355	3301
GSE75330	mouse	fluidigm c1	Directed linear	clustering: BackSpinV2	3694	999
		fluidigm c1	Multifurcation	clustering: BackSpinV2	4959	1008
GSE85066	human	fluidigm c1	Rooted tree	FACS	501	3523
		SC3-seq	Rooted tree	clustering: hierarchical	182	5273
GSE74767	macaque	SC3-seq	Directed linear	clustering: hierarchical	83	4780
		SC3-seq	Rooted tree	clustering: hierarchical	272	5376
		SC3-seq	Disconnected directed graph	clustering: hierarchical	351	5416
GSE70240, GSE70243, GSE70244, GSE70236	mouse	fluidigm c1	Directed linear	FACS	318	2702
GSE70240, GSE70244, GSE70236	mouse	fluidigm c1	Rooted tree	FACS, clustering: ICGS	376	2967
GSE79363	mouse	smart-seq2	Bifurcation	FACS	60	1849
		fluidigm c1	Directed linear	clustering: AP	749	1033
GSE67602	mouse	fluidigm c1	Directed linear	clustering: AP	699	1060
		fluidigm c1	Directed linear	clustering: AP	346	1009
GSE90860	mouse	fluidigm c1	Bifurcation	timeseries, clustering: PCA + hierarchical	213	4872
		smart-seq2	Directed linear	clustering: DBclust	659	3630
GSE86146	human	smart-seq2	Directed linear	clustering: DBclust	629	4691
		smart-seq2	Directed linear	timeseries, sample origin	659	3630
		smart-seq2	Directed linear	timeseries, sample origin	671	4459
GSE87375	mouse	smart-seq2	Directed linear	timeseries	322	6763
		smart-seq2	Directed linear	timeseries	563	6372
GSE90047	mouse	smart-seq2	Bifurcation	timeseries,clustering: PCA + hierarchical	503	2037
GSE99951	mouse	smart-seq2	Directed linear	timeseries	192	1534
		smart-seq2	Directed linear	timeseries	456	3585
		smart-seq2	Directed linear	timeseries, clustering: ICIM	277	1514
GSE100058	fly	smart-seq2	Directed acyclic graph	timeseries, clustering: ICIM	169	982
		smart-seq2	Disconnected directed graph	timeseries, clustering: ICIM	454	2703
GSE95315	mouse	10X chromium	Directed linear	clustering: markov	3580	698
GSE98664	mouse	rdRNA-seq	Directed linear	timeseries	414	12532

Supplementary Table 2 Importance values for algorithmic components. The overall performance was compared between methods containing a particular algorithmic component, and we calculated both a corrected p-value using a two-tailed Mann–Whitney U test and the increase in node purity importance measure using random forest classification.

Component	Adjusted p-value	Inc node purity	Effect	Methods	Category
principal curves	0.08842	0.213	Higher performance	SCUBA, Embeddr, SLICE, SCORPIUS, Slingshot	ordering
k-means	0.08842	0.557	Higher performance	Waterfall, SCORPIUS, Slingshot, Monocle DDRTree, reCAT	clustering
any graph building	0.08842	0.739	Higher performance	Monocle ICA, Waterfall, Wishbone, StemID, TSCAN, SLICER, Mpath, cellTree Maptpx, cellTree Gibbs, cellTree VEM, SLICE, Topslam, SCORPIUS, Slingshot, Monocle DDRTree, Sincell, Wanderlust, reCAT	graph building
MST	0.13634	0.478	Higher performance	Monocle ICA, Waterfall, StemID, TSCAN, Mpath, cellTree Maptpx, cellTree Gibbs, cellTree VEM, SLICE, Topslam, Slingshot, Monocle DDRTree, Sincell	graph building
tSNE	0.13634	0.442	Lower performance	StemID, Pseudogp, Topslam, Sincell	dimensionality reduction
any ordering	0.13634	0.158	Higher performance	Monocle ICA, SCUBA, StemID, TSCAN, SLICER, Embeddr, SLICE, SCORPIUS, Slingshot, Monocle DDRTree, Wanderlust	ordering
any clustering	0.13634	0.113	Higher performance	Waterfall, SCUBA, StemID, TSCAN, Mpath, SLICE, SCORPIUS, Slingshot, Monocle DDRTree, Sincell, reCAT	clustering
ICA	0.31683	0.059	Lower performance	Monocle ICA, Pseudogp, Topslam, Sincell	dimensionality reduction
any dimensionality reduction	0.33783	0.251	Higher performance	Monocle ICA, Waterfall, Wishbone, DPT, StemID, TSCAN, SLICER, Pseudogp, Embeddr, SLICE, Topslam, SCORPIUS, Slingshot, Monocle DDRTree, Sincell	dimensionality reduction
any dimensionality reduction	0.33783	0.251	Higher performance	Monocle ICA, Waterfall, Wishbone, DPT, StemID, TSCAN, SLICER, Pseudogp, Embeddr, SLICE, Topslam, SCORPIUS, Slingshot, Monocle DDRTree, Sincell	dimensionality reduction
knn graph	0.39281	0.154	Lower performance	Wishbone, SLICER, Sincell, Wanderlust	graph building
MDS	0.61356	0.074	Lower performance	Pseudogp, Topslam, SCORPIUS, Sincell	dimensionality reduction
any generative model	0.73132	0.079	Lower performance	GPFates, Pseudogp, SCOUP, cellTree Maptpx, cellTree Gibbs, cellTree VEM, MFA, Phenopath, SCIMITAR	generative model
PCA	0.9791	0.033	Lower performance	Waterfall, TSCAN, Pseudogp, SLICE, Topslam, Sincell	dimensionality reduction
k-medoids	0.9791	0.057	Lower performance	StemID, SLICE, Sincell, reCAT	clustering

Supplementary Table 3 Scoring scheme for method quality control. Each quality aspect was given a weight based on how many times it was mentioned in a set of articles discussing best practices for tool development.

Category	Aspect	Weight	References	Item	Item weight
Availability	Is the code freely available?	8	[71, 72, 92, 93, 94, 95, 96]	Code is freely available	0.50
				Tool can be run on a freely available platform	0.50
	Is the code version controlled?	7	[71, 72, 92, 93, 94, 95]	Code is available on a public version controlled repository, such as Github	1.00
	Is the code contained into an easy to install package? Is it discoverable?	5	[92, 94, 95, 96]	The code is provided as a "package", exposing functionality through functions or shell commands	0.50
				The code can be easily installed through a repository such as CRAN, Bioconductor, PyPI, CPAN, debian packages, ...	0.50
	Are dependencies clearly stated and available?	5	[72, 93, 94, 97]	Dependencies are clearly stated in the tutorial or in the code	0.50
				Dependencies are automatically installed	0.50
	Is the license of the code clear? Does this license permit academic use?	7	[72, 92, 93, 94, 95, 96]	License of the code is clear	0.50
				License allows academic use	0.50
	Does the tool have a graphical user interface?	2	[95]	The tool can be run using a graphical user interface, either locally or on a web server	0.50
				The tool can be run through the command line or through a programming language	0.50
Code quality	Do the functions and objects have a consistent naming?	3	[71, 94]	Functions have well chosen names	0.67
				Arguments have well chosen names	0.33
	Do the functions have a consistent style?	4	[71, 72, 94]	Code is styled consistently	0.50
				Code following (basic) good practices in the programming language of choice	0.50
	Is code frequently duplicated?	3	[71, 94]	Duplicated code is minimal	1.00
	Does the code expose certain steps in the method as self-contained functions or commands?	4	[93, 95, 98]	The method is exposed to the user as self-contained functions or commands	1.00
	Does the package allow plotting of (intermediate) results?	1		Plotting functions are provided for the final and/or intermediate results	1.00
	Does the package include dummy proofing?	3	[92, 97]	Package contains dummy proofing, i.e. testing whether the parameters and data supplied by the user make sense and are useful	1.00
Code assurance	Does the package include some testing?	6	[71, 92, 94, 95, 98]	Code contains unit tests	0.50
				Tests are run automatically using functionality from the programming language	0.50
	Does the package include some continuous integration?	5	[72, 94, 95, 99]	Continuous integration, for example on Travis CI	1.00
	Does the package include a system to ask for support?	6	[71, 72, 94, 95, 96]	There is a support ticket system, for example on Github	0.50
				Tickets are closed within a reasonable time frame	0.50
Documentation	Is there a tutorial available for the method? Does this tutorial show everything the user needs?	6	[94, 95, 96, 97, 100]	A tutorial or vignette is available	0.25
				The tutorial has example results	0.25
				The tutorial has real example data	0.25
				The use of the method is shown with several (1=0, 2=0.5, >2=1) datasets	0.25
	Is the purpose and usage of each function documented?	6	[71, 93, 94, 95, 97]	The purpose and usage of functions is documented	0.33
				The parameters of functions are documented	0.33
				The output of functions is documented	0.33
	Is the code documented inline?	6		Inline documentation is present in the code, to make it understandable	1.00
	Are all important parameters available to the user?	2	[93]	All important parameters are exposed to the user	1.00
Behaviour	Are no seeds set during the execution of the method?	2	[101]	No seeds (1), some seeds (0.5) or a lot of seeds (0) are set	1.00
	Is unexpected output generated by the method?	2	[72]	No unexpected output messages	0.25
				No unexpected files, folders or plots are generated	0.25
				No unexpected warnings during runtime or compilation are generated	0.50
	Was postprocessing necessary to get the output of the method into a useful format?	1		The postprocessing is minimal (1), moderate (0.5) or extensive (0)	1.00
	How fast can the method run on default parameters?	1		Using the average execution times on benchmark datasets; options: <1m (1), <1h (.5), >1h (0)	1.00
	Does the method require prior information?	1		Prior information is required (0), optional (1) or not required (1)	1.00
Paper	Is the method published?	1		The method is published	1.00
	Is the method published in a peer-reviewed journal?	4	[97, 102, 103]	The paper is published in a peer-reviewed journal	1.00
	Is the methods usefulness shown in the paper?	3	[104, 105]	The paper shows the method's usefulness on several (1), one (0.25) or no datasets.	0.50
				The paper quantifies the accuracy of the method on real data given a gold or silver standard trajectory	0.50
	Does the paper assess method robustness?	5	[97, 100, 104, 105]	Method robustness (to eg. noise, subsampling, parameter changes, stability) is assessed in one (0.5) or several (1) ways	1.00

Supplementary Table 4 Distributions from which each parameter in the thermodynamic model was sampled.

$r = \mathcal{U}(10, 200)$
$d = \mathcal{U}(2, 8)$
$p = \mathcal{U}(2, 8)$
$q = \mathcal{U}(1, 5)$
$a0 = \begin{cases} 1 & \text{if } e \in \{1\} \text{ or } e = \emptyset \\ 0 & \text{if } e \in \{-1\} \\ 0.5 & \text{otherwise} \end{cases}$
$a_i = \begin{cases} 1 & \text{if } e_i \in \{-1\} \\ 0 & \text{otherwise} \end{cases}$
$s = \mathcal{U}(1, 20)$
$k = y_{max}/(2 * s)$
$c = \mathcal{U}(1, 4)$
where
$y_{max} = r/d * p/q$

Colophon

This report was generated on 2018-03-05 23:36:32 using R version 3.4.3 (2017-11-30) and the following packages:

package	*	version	date	source
acepack		1.4.1	2016-10-29	CRAN (R 3.4.1)
AnnotationDbi		1.40.0	2018-01-11	cran (**1.40.0**)
ape		5.0	2017-10-30	cran (**5.0**)
aroma.light		3.8.0	2018-01-11	cran (**3.8.0**)
assertthat		0.2.0	2017-04-11	CRAN (R 3.4.1)
backports		1.1.2	2017-12-13	cran (**1.1.2**)
base	*	3.4.3	2017-12-01	local
base64enc		0.1-3	2015-07-28	CRAN (R 3.4.1)
BBmisc		1.11	2017-03-10	cran (**1.11**)
beeswarm		0.2.3	2016-04-25	CRAN (R 3.4.1)
bindr		0.1	2016-11-13	CRAN (R 3.4.1)
bindrcpp	*	0.2	2017-06-17	CRAN (R 3.4.1)
Biobase		2.38.0	2018-01-11	cran (**2.38.0**)
BiocGenerics		0.24.0	2018-01-11	cran (**0.24.0**)
BiocParallel		1.12.0	2018-01-11	cran (**1.12.0**)
biomaRt		2.34.2	2018-02-02	cran (**2.34.2**)
bit		1.1-12	2014-04-09	CRAN (R 3.4.1)
bit64		0.9-7	2017-05-08	CRAN (R 3.4.1)
bitops		1.0-6	2013-08-17	CRAN (R 3.4.1)
blob		1.1.0	2017-06-17	CRAN (R 3.4.1)
broom		0.4.3	2017-11-20	cran (**0.4.3**)
caret		6.0-78	2017-12-10	cran (**6.0-78**)
caTools		1.17.1	2014-09-10	cran (**1.17.1**)
cellranger		1.1.0	2016-07-27	CRAN (R 3.4.1)
checkmate		1.8.5	2017-10-24	CRAN (R 3.4.2)
class		7.3-14	2015-08-30	CRAN (R 3.4.0)
cli		1.0.0	2017-11-05	cran (**1.0.0**)
cluster		2.0.6	2017-03-16	CRAN (R 3.4.0)
codetools		0.2-15	2016-10-05	CRAN (R 3.3.1)

package	*	version	date	source
colorspace		1.3-2	2016-12-14	CRAN (R 3.4.1)
compiler		3.4.3	2017-12-01	local
cowplot	*	0.9.2	2017-12-17	cran (**0.9.2**)
crayon		1.3.4	2017-09-16	CRAN (R 3.4.1)
curl		3.1	2017-12-12	cran (**3.1**)
CVST		0.2-1	2013-12-10	CRAN (R 3.4.1)
data.table		1.10.4-3	2017-10-27	CRAN (R 3.4.2)
datasets	*	3.4.3	2017-12-01	local
DBI		0.7	2017-06-18	CRAN (R 3.4.1)
ddalpha		1.3.1.1	2018-02-02	cran (**1.3.1.1**)
DelayedArray		0.4.1	2018-01-11	cran (**0.4.1**)
DEoptimR		1.0-8	2016-11-19	cran (**1.0-8**)
devtools		1.13.4	2017-11-09	cran (**1.13.4**)
diffusionMap		1.1-0	2014-02-20	cran (**1.1-0**)
digest		0.6.15	2018-01-28	cran (**0.6.15**)
dimRed		0.1.0	2017-05-04	CRAN (R 3.4.1)
diptest		0.75-7	2016-12-05	cran (**0.75-7**)
dplyr	*	0.7.4	2017-09-28	CRAN (R 3.4.3)
DRR		0.0.2	2016-09-15	CRAN (R 3.4.1)
DT		0.4	2018-01-30	cran (**0.4**)
dtw		1.18-1	2015-09-01	CRAN (R 3.4.1)
dynalysis	*	0.0.0.9000	2018-03-02	local
dynamicTreeCut		1.63-1	2016-03-11	cran (**1.63-1**)
dyneval	*	0.1.0	2018-02-21	local
dyngen	*	0.1	2018-02-06	local
dynmethods	*	0.1.0	2018-02-26	Github (rcannood/dynmethods@89eb5d1)
dynnormaliser		0.1.0	2018-02-26	Github (rcannood/dynnormaliser@3fc158a)
dynplot		0.1.0	2018-02-26	Github (Zouter/dynplot@d918de5)
dyntoy	*	0.1.0	2018-02-26	Github (Zouter/dyntoy@f6b87cb)
dynutils	*	0.1.0	2018-03-02	local
edgeR		3.20.8	2018-02-21	cran (**3.20.8**)
evaluate		0.10.1	2017-06-24	CRAN (R 3.4.1)
fastgssa		0.6-0	2018-01-11	Github (dynverse/fastgssa@92cfc08)
fitdistrplus		1.0-9	2017-03-24	cran (**1.0-9**)
flexmix		2.3-14	2017-04-28	cran (**2.3-14**)
FNN		1.1	2013-07-31	cran (**1.1**)
forcats	*	0.2.0	2017-01-23	CRAN (R 3.4.1)
foreach		1.4.4	2017-12-12	cran (**1.4.4**)
foreign		0.8-69	2017-06-21	CRAN (R 3.4.0)
Formula		1.2-2	2017-07-10	CRAN (R 3.4.1)
fpc		2.1-11	2018-01-13	cran (**2.1-11**)
GA		3.0.2	2016-06-07	CRAN (R 3.4.1)
gdata		2.18.0	2017-06-06	CRAN (R 3.4.1)
GenomeInfoDb		1.14.0	2018-01-11	cran (**1.14.0**)
GenomeInfoDbData		1.0.0	2018-01-11	cran (**1.0.0**)
GenomicRanges		1.30.1	2018-01-11	cran (**1.30.1**)
ggbeeswarm		0.6.0	2017-08-07	CRAN (R 3.4.1)
ggforce		0.1.1	2016-11-28	cran (**0.1.1**)
ggplot2	*	2.2.1	2016-12-30	CRAN (R 3.4.1)
ggraph		1.0.1	2018-01-29	cran (**1.0.1**)
ggrepel		0.7.0	2017-09-29	cran (**0.7.0**)
ggridges		0.4.1	2017-09-15	cran (**0.4.1**)

package	*	version	date	source
glue		1.2.0	2017-10-29	CRAN (R 3.4.2)
googledrive	*	0.1.1	2017-08-28	CRAN (R 3.4.3)
gower		0.1.2	2017-02-23	CRAN (R 3.4.1)
gplots		3.0.1	2016-03-30	cran (**3.0.1**)
graphics	*	3.4.3	2017-12-01	local
grDevices	*	3.4.3	2017-12-01	local
grid		3.4.3	2017-12-01	local
gridExtra		2.3	2017-09-09	CRAN (R 3.4.1)
gtable		0.2.0	2016-02-26	CRAN (R 3.4.1)
gtools		3.5.0	2015-05-29	CRAN (R 3.4.1)
haven		1.1.1	2018-01-18	CRAN (R 3.4.3)
Hmisc		4.1-1	2018-01-03	cran (**4.1-1**)
hms		0.4.1	2018-01-24	CRAN (R 3.4.3)
htmlTable		1.11.1	2017-12-27	cran (**1.11.1**)
htmltools		0.3.6	2017-04-28	CRAN (R 3.4.1)
htmlwidgets		1.0	2018-01-20	cran (**1.0**)
httpuv		1.3.5	2017-07-04	CRAN (R 3.4.1)
httr		1.3.1	2017-08-20	cran (**1.3.1**)
ica		1.0-1	2015-08-25	cran (**1.0-1**)
igraph		1.1.0	2018-02-02	Github (igraph/rigraph@057cc9d)
ipred		0.9-6	2017-03-01	CRAN (R 3.4.1)
IRanges		2.12.0	2018-01-11	cran (**2.12.0**)
irlba		2.3.2	2018-01-11	cran (**2.3.2**)
iterators		1.0.9	2017-12-12	cran (**1.0.9**)
jsonlite		1.5	2017-06-01	cran (**1.5**)
kernlab		0.9-25	2016-10-03	cran (**0.9-25**)
KernSmooth		2.23-15	2015-06-29	CRAN (R 3.4.0)
knitr		1.19	2018-01-29	CRAN (R 3.4.3)
lars		1.2	2013-04-24	cran (**1.2**)
lattice		0.20-35	2017-03-25	CRAN (R 3.3.3)
latticeExtra		0.6-28	2016-02-09	CRAN (R 3.4.1)
lava		1.5.1	2017-09-27	CRAN (R 3.4.1)
lazyeval		0.2.1	2017-10-29	cran (**0.2.1**)
lhs		0.16	2018-01-04	cran (**0.16**)
limma		3.34.9	2018-02-26	cran (**3.34.9**)
limSolve		1.5.5.3	2017-08-14	cran (**1.5.5.3**)
locfit		1.5-9.1	2013-04-20	CRAN (R 3.4.1)
lpSolve		5.6.13	2015-09-19	cran (**5.6.13**)
lubridate		1.7.2	2018-02-06	CRAN (R 3.4.3)
magrittr		1.5	2014-11-22	CRAN (R 3.4.1)
MASS		7.3-49	2018-02-23	CRAN (R 3.4.3)
Matrix		1.2-11	2017-08-16	CRAN (R 3.4.1)
matrixStats		0.53.1	2018-02-11	cran (**0.53.1**)
mclust		5.4	2017-11-22	cran (**5.4**)
mco		1.0-15.1	2014-11-29	cran (**1.0-15** .)
memoise		1.1.0	2017-04-21	CRAN (R 3.4.1)
metap		0.8	2017-01-22	cran (**0.8**)
methods	*	3.4.3	2017-12-01	local
mime		0.5	2016-07-07	CRAN (R 3.4.1)
misc3d		0.8-4	2013-01-25	cran (**0.8-4**)
mixtools		1.1.0	2017-03-10	cran (**1.1.0**)
mnormt		1.5-5	2016-10-15	CRAN (R 3.4.1)

package	*	version	date	source
ModelMetrics		1.1.0	2016-08-26	CRAN (R 3.4.1)
modelr		0.1.1	2017-07-24	CRAN (R 3.4.1)
modeltools		0.2-21	2013-09-02	CRAN (R 3.4.1)
munsell		0.4.3	2016-02-13	CRAN (R 3.4.1)
mvtnorm		1.0-7	2018-01-26	cran (**1.0-7**)
nlme		3.1-131	2017-02-06	CRAN (R 3.4.0)
nnet		7.3-12	2016-02-02	CRAN (R 3.4.0)
numDeriv		2016.8-1	2016-08-27	cran (**2016.8** -)
openssl		0.9.9	2017-11-10	cran (**0.9.9**)
parallel		3.4.3	2017-12-01	local
ParamHelpers		1.11	2018-01-16	Github (berndbischl/ParamHelpers@59c649e)
pbapply		1.3-4	2018-01-10	cran (**1.3-4**)
pdist		1.2	2013-02-03	CRAN (R 3.4.1)
pheatmap		1.0.8	2015-12-11	CRAN (R 3.4.1)
pillar		1.1.0	2018-01-14	cran (**1.1.0**)
pkgconfig		2.0.1	2017-03-21	CRAN (R 3.4.1)
plot3D		1.1.1	2017-08-28	cran (**1.1.1**)
plotly		4.7.1	2017-07-29	cran (**4.7.1**)
plyr		1.8.4	2016-06-08	CRAN (R 3.4.1)
prabclus		2.2-6	2015-01-14	cran (**2.2-6**)
prettyunits		1.0.2	2015-07-13	cran (**1.0.2**)
princurve		1.1-12	2013-04-25	cran (**1.1-12**)
PRISM		1.0	2018-01-11	Github (rcannood/PRISM@255f82b)
prodlim		1.6.1	2017-03-06	CRAN (R 3.4.1)
progress		1.1.2	2016-12-14	cran (**1.1.2**)
proxy		0.4-21	2018-01-04	cran (**0.4-21**)
psych		1.7.8	2017-09-09	CRAN (R 3.4.1)
purrr	*	0.2.4	2017-10-18	cran (**0.2.4**)
quadprog		1.5-5	2013-04-17	cran (**1.5-5**)
R.methodsS3		1.7.1	2016-02-16	cran (**1.7.1**)
R.oo		1.21.0	2016-11-01	cran (**1.21.0**)
R.utils		2.6.0	2017-11-05	cran (**2.6.0**)
R6		2.2.2	2017-06-17	CRAN (R 3.4.1)
random		0.2.6	2017-02-05	cran (**0.2.6**)
randomForest		4.6-12	2015-10-07	CRAN (R 3.4.1)
ranger		0.9.0	2018-01-09	cran (**0.9.0**)
RColorBrewer		1.1-2	2014-12-07	CRAN (R 3.4.1)
Rcpp		0.12.15	2018-01-20	CRAN (R 3.4.3)
RcppRoll		0.2.2	2015-04-05	CRAN (R 3.4.3)
RCurl		1.95-4.10	2018-01-04	cran (**1.95-4** .)
readr	*	1.1.1	2017-05-16	CRAN (R 3.4.1)
readxl		1.0.0	2017-04-18	CRAN (R 3.4.1)
recipes		0.1.2	2018-01-11	cran (**0.1.2**)
reshape2		1.4.3	2017-12-11	cran (**1.4.3**)
rhdf5		2.22.0	2018-02-16	Bioconductor
rjson		0.2.15	2014-11-03	CRAN (R 3.4.1)
RJSONIO		1.3-0	2014-07-28	cran (**1.3-0**)
rlang		0.2.0	2018-02-20	cran (**0.2.0**)
rmarkdown	*	1.8	2017-11-17	CRAN (R 3.4.3)
robustbase		0.92-8	2017-11-01	cran (**0.92-8**)
ROCR		1.0-7	2015-03-26	cran (**1.0-7**)
rpart		4.1-13	2018-02-23	CRAN (R 3.4.3)

package	*	version	date	source
rprojroot		1.3-2	2018-01-03	cran (**1.3-2**)
RSQLite		2.0	2017-06-19	CRAN (R 3.4.1)
rstudioapi		0.7	2017-09-07	CRAN (R 3.4.1)
Rtsne		0.13	2017-04-14	CRAN (R 3.4.1)
rvest		0.3.2	2016-06-17	CRAN (R 3.4.1)
S4Vectors		0.16.0	2018-01-11	cran (**0.16.0**)
scales		0.5.0	2017-08-24	CRAN (R 3.4.1)
scater		1.6.3	2018-02-21	cran (**1.6.3**)
scatterplot3d		0.3-40	2017-04-22	cran (**0.3-40**)
SCORPIUS		1.0	2018-01-11	Github (rcannood/SCORPIUS@6e4d1a5)
scran		1.6.8	2018-02-26	cran (**1.6.8**)
SDMTools		1.1-221	2014-08-05	cran (**1.1-221**)
segmented		0.5-3.0	2017-11-30	cran (**0.5-3.0**)
Seurat		2.2.1	2018-02-14	cran (**2.2.1**)
sfsmisc		1.1-1	2017-06-08	CRAN (R 3.4.1)
shiny		1.0.5	2017-08-23	CRAN (R 3.4.1)
shinydashboard		0.6.1	2017-06-14	CRAN (R 3.4.1)
SingleCellExperiment		1.0.0	2018-01-11	cran (**1.0.0**)
smoof		1.5.1	2017-08-14	cran (**1.5.1**)
sn		1.5-1	2017-11-23	cran (**1.5-1**)
splines		3.4.3	2017-12-01	local
statmod		1.4.30	2017-06-18	cran (**1.4.30**)
stats	*	3.4.3	2017-12-01	local
stats4		3.4.3	2017-12-01	local
stringi		1.1.6	2017-11-17	cran (**1.1.6**)
stringr	*	1.3.0	2018-02-19	cran (**1.3.0**)
SummarizedExperiment		1.8.1	2018-01-11	cran (**1.8.1**)
survival		2.41-3	2017-04-04	CRAN (R 3.4.0)
tclust		1.3-1	2017-08-24	cran (**1.3-1**)
testthat		2.0.0	2017-12-13	cran (**2.0.0**)
tibble	*	1.4.2	2018-01-22	cran (**1.4.2**)
tidygraph		1.1.0	2018-02-10	cran (**1.1.0**)
tidyr	*	0.8.0	2018-01-29	cran (**0.8.0**)
tidyselect		0.2.3	2017-11-06	cran (**0.2.3**)
tidyverse	*	1.2.1	2017-11-14	cran (**1.2.1**)
timeDate		3042.101	2017-11-16	cran (**3042.10**)
tools		3.4.3	2017-12-01	local
trimcluster		0.1-2	2012-10-29	cran (**0.1-2**)
tsne		0.1-3	2016-07-15	cran (**0.1-3**)
TSP		1.1-5	2017-02-22	cran (**1.1-5**)
tweenr		0.1.5	2016-10-10	cran (**0.1.5**)
tximport		1.6.0	2018-01-11	cran (**1.6.0**)
udunits2		0.13	2016-11-17	cran (**0.13**)
units		0.5-1	2018-01-08	cran (**0.5-1**)
utils	*	3.4.3	2017-12-01	local
VGAM		1.0-5	2018-02-07	cran (**1.0-5**)
vipor		0.4.5	2017-03-22	CRAN (R 3.4.1)
viridis		0.5.0	2018-02-02	cran (**0.5.0**)
viridisLite		0.3.0	2018-02-01	cran (**0.3.0**)
withr		2.1.1	2017-12-19	cran (**2.1.1**)
XML		3.98-1.9	2017-06-19	CRAN (R 3.4.1)
xml2		1.2.0	2018-01-24	cran (**1.2.0**)

package	*	version	date	source
xtable		1.8-2	2016-02-05	CRAN (R 3.4.1)
XVector		0.18.0	2018-01-11	cran (**0.18.0**)
yaml		2.1.16	2017-12-12	cran (**2.1.16**)
zlibbioc		1.24.0	2018-01-11	cran (**1.24.0**)
zoo		1.8-1	2018-01-08	cran (**1.8-1**)

References

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

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

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

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

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

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

[7] L. Velten et al. "Human Haematopoietic Stem Cell Lineage Commitment Is a Continuous Process., Human Haematopoietic Stem Cell Lineage Commitment Is a Continuous Process". eng. In: *Nature cell biology, Nature cell biology* 19, 19.4, 4 (Apr. 2017), pp. 271, 271–281. ISSN: 1465-7392. DOI: 10.1038/ncb3493,\%002010.1038/ncb3493.

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

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

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

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

[12] Philipp Angerer et al. "Single Cells Make Big Data: New Challenges and Opportunities in Transcriptomics". In: *Current Opinion in Systems Biology*. Big data acquisition and analysis • Pharmacology and drug discovery 4 (Aug. 2017), pp. 85–91. ISSN: 2452-3100. DOI: 10.1016/j.coisb.2017.07.004.

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

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

[15] Eugenio Marco et al. "Bifurcation Analysis of Single-Cell Gene Expression Data Reveals Epigenetic Landscape". en. In: *Proceedings of the National Academy of Sciences* 111.52 (Dec. 2014), E5643–E5650. ISSN: 0027-8424, 1091-6490. DOI: 10.1073/pnas.1408993111.

[16] Miguel Juliá, Amalio Telenti, and Antonio Rausell. "Sincell: An R/Bioconductor Package for Statistical Assessment of Cell-State Hierarchies from Single-Cell RNA-Seq". en. In: *Bioinformatics* 31.20 (Oct. 2015), pp. 3380–3382. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btv368.

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

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

[19] Kieran Campbell, Chris P. Ponting, and Caleb Webber. "Laplacian Eigenmaps and Principal Curves for High Resolution Pseudotemporal Ordering of Single-Cell RNA-Seq Profiles". en. In: *bioRxiv* (Sept. 2015), p. 027219. DOI: 10.1101/027219.

[20] Gregory Giecold et al. "Robust Lineage Reconstruction from High-Dimensional Single-Cell Data". en. In: *Nucleic Acids Research* 44.14 (Aug. 2016), e122–e122. ISSN: 0305-1048. DOI: 10.1093/nar/gkw452.

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

[22] Kieran R. Campbell and Christopher Yau. "Order Under Uncertainty: Robust Differential Expression Analysis Using Probabilistic Models for Pseudotime Inference". en. In: *PLOS Computational Biology* 12.11 (Nov. 2016), e1005212. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1005212.

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

[24] Aaron Diaz et al. "SCell: Integrated Analysis of Single-Cell RNA-Seq Data". en. In: *Bioinformatics* 32.14 (July 2016), pp. 2219–2220. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btw201.

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

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

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

[28] John E. Reid and Lorenz Wernisch. "Pseudotime Estimation: Deconfounding Single Cell Time Series". en. In: *Bioinformatics* 32.19 (Oct. 2016), pp. 2973–2980. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btw372.

[29] Dominic Grün et al. "De Novo Prediction of Stem Cell Identity Using Single-Cell Transcriptome Data". In: *Cell Stem Cell* 19.2 (Aug. 2016), pp. 266–277. ISSN: 1934-5909. DOI: 10.1016/j.stem.2016.05.010.

[30] Kieran R. Campbell and Christopher Yau. "A Descriptive Marker Gene Approach to Single-Cell Pseudotime Inference". en. In: *bioRxiv* (Nov. 2017), p. 060442. DOI: 10.1101/060442.

[31] Jinmiao Chen et al. "Mpath Maps Multi-Branching Single-Cell Trajectories Revealing Progenitor Cell Progression during Development". en. In: *Nature Communications* 7 (June 2016), p. 11988. ISSN: 2041-1723. DOI: 10.1038/ncomms11988.

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

[33] Li-Fang Chu et al. "Single-Cell RNA-Seq Reveals Novel Regulators of Human Embryonic Stem Cell Differentiation to Definitive Endoderm". In: *Genome Biology* 17 (Aug. 2016), p. 173. ISSN: 1474-760X. DOI: 10.1186/s13059-016-1033-x.

[34] Pablo Cordero and Joshua M. Stuart. "TRACING CO-REGULATORY NETWORK DYNAMICS IN NOISY, SINGLE-CELL TRANSCRIPTOME TRAJECTORIES". eng. In: *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing* 22 (2017), pp. 576–587. ISSN: 2335-6936. DOI: 10.1142/9789813207813_0053.

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

[36] Andrew E. Teschendorff and Tariq Enver. "Single-Cell Entropy for Accurate Estimation of Differentiation Potency from a Cell's Transcriptome". en. In: *Nature Communications* 8 (June 2017), p. 15599. ISSN: 2041-1723. DOI: 10.1038/ncomms15599.

[37] Nikolaos K. Chlis, F. Alexander Wolf, and Fabian J. Theis. "Model-Based Branching Point Detection in Single-Cell Data by K-Branches Clustering". en. In: *Bioinformatics* 33.20 (Oct. 2017), pp. 3211–3219. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btx325.

[38] Minzhe Guo et al. "SLICE: Determining Cell Differentiation and Lineage Based on Single Cell Entropy". en. In: *Nucleic Acids Research* 45.7 (Apr. 2017), e54–e54. ISSN: 0305-1048. DOI: 10.1093/nar/gkw1278.

[39] Max Zwiessele and Neil D. Lawrence. "Topslam: Waddington Landscape Recovery for Single Cell Experiments". en. In: *bioRxiv* (Feb. 2017), p. 057778. DOI: 10.1101/057778.

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

[41] Xun Zhu et al. "Granatum: A Graphical Single-Cell RNA-Seq Analysis Pipeline for Genomics Scientists". In: *Genome Medicine* 9 (Dec. 2017), p. 108. ISSN: 1756-994X. DOI: 10.1186/s13073-017-0492-3.

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

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

[44] Kevin R. Moon et al. "Visualizing Transitions and Structure for High Dimensional Data Exploration". en. In: *bioRxiv* (Dec. 2017), p. 120378. DOI: 10.1101/120378.

[45] Sabrina Rashid, Darrell N. Kotton, and Ziv Bar-Joseph. "TASIC: Determining Branching Models from Time Series Single Cell Data". en. In: *Bioinformatics* 33.16 (Aug. 2017), pp. 2504–2512. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btx173.

[46] Tao Peng and Qing Nie. "SOMSC: Self-Organization-Map for High-Dimensional Single-Cell Data of Cellular States and Their Transitions". en. In: *bioRxiv* (Aug. 2017), p. 124693. DOI: 10.1101/124693.

[47] Kelly Street et al. "Slingshot: Cell Lineage and Pseudotime Inference for Single-Cell Transcriptomics". en. In: *bioRxiv* (Apr. 2017), p. 128843. DOI: 10.1101/128843.

[48] Abbas H. Rizvi et al. "Single-Cell Topological RNA-Seq Analysis Reveals Insights into Cellular Differentiation and Development". en. In: *Nature Biotechnology* 35.6 (June 2017), pp. 551–560. ISSN: 1546-1696. DOI: 10.1038/nbt.3854.

[49] *Prior Knowledge And Sampling Model Informed Learning With Single Cell RNA-Seq Data | bioRxiv*. https://www.biorxiv.org/content/early/2017/05/

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

[51] Mayank Sharma et al. "FORKS: Finding Orderings Robustly Using K-Means and Steiner Trees". en. In: *bioRxiv* (June 2017), p. 132811. DOI: 10.1101/132811.

[52] Joshua D. Welch, Alexander J. Hartemink, and Jan F. Prins. "MATCHER: Manifold Alignment Reveals Correspondence between Single Cell Transcriptome and Epigenome Dynamics". In: *Genome Biology* 18 (July 2017), p. 138. ISSN: 1474-760X. DOI: 10.1186/s13059-017-1269-0.

[53] Kieran Campbell and Christopher Yau. "Uncovering Genomic Trajectories with Heterogeneous Genetic and Environmental Backgrounds across Single-Cells and Populations". en. In: *bioRxiv* (July 2017), p. 159913. DOI: 10.1101/159913.

[54] Jing Guo and Jie Zheng. "HopLand: Single-Cell Pseudotime Recovery Using Continuous Hopfield Network-Based Modeling of Waddington's Epigenetic Landscape". en. In: *Bioinformatics* 33.14 (July 2017), pp. i102–i109. ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btx232.

[55] Shuxiong Wang, Adam MacLean, and Qing Nie. "Low-Rank Similarity Matrix Optimization Identifies Subpopulation Structure and Orders Single Cells in Pseudotime". en. In: *bioRxiv* (July 2017), p. 168922. DOI: 10.1101/168922.

[56] Caleb Weinreb et al. "Fundamental Limits on Dynamic Inference from Single-Cell Snapshots". en. In: *Proceedings of the National Academy of Sciences* (Feb. 2018), p. 201714723. ISSN: 0027-8424, 1091-6490. DOI: 10.1073/pnas.1714723115.

[57] Alexis Boukouvalas, James Hensman, and Magnus Rattray. "BGP: Branched Gaussian Processes for Identifying Gene-Specific Branching Dynamics in Single Cell Data". en. In: *bioRxiv* (Aug. 2017), p. 166868. DOI: 10.1101/166868.

[58] F. Alexander Wolf, Philipp Angerer, and Fabian J. Theis. "SCANPY: Large-Scale Single-Cell Gene Expression Data Analysis". In: *Genome Biology* 19 (Feb. 2018), p. 15. ISSN: 1474-760X. DOI: 10.1186/s13059-017-1382-0.

[59] Christopher Andrew Penfold et al. "Nonparametric Bayesian Inference of Transcriptional Branching and Recombination Identifies Regulators of Early Human Germ Cell Development". en. In: *bioRxiv* (Sept. 2017), p. 167684. DOI: 10.1101/167684.

[60] Geoffrey Schiebinger et al. "Reconstruction of Developmental Landscapes by Optimal-Transport Analysis of Single-Cell Gene Expression Sheds Light on Cellular Reprogramming." en. In: *bioRxiv* (Sept. 2017), p. 191056. DOI: 10.1101/191056.

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

[62] Magdalena E. Strauss, John E. Reid, and Lorenz Wernisch. "GPseudoRank: A Permutation Sampler for Single Cell Orderings". en. In: *bioRxiv* (Feb. 2018), p. 211417. DOI: 10.1101/211417.

[63] Charles A. Herring et al. "Unsupervised Trajectory Analysis of Single-Cell RNA-Seq and Imaging Data Reveals Alternative Tuft Cell Origins in the Gut". English. In: *Cell Systems* 6.1 (Jan. 2018), 37–51.e9. ISSN: 2405-4712. DOI: 10.1016/j.cels.2017.10.012.

[64] Na Sun et al. "Inference of Differentiation Time for Single Cell Transcriptomes Using Cell Population Reference Data". en. In: *Nature Communications* 8.1 (Nov. 2017), p. 1856. ISSN: 2041-1723. DOI: 10.1038/s41467-017-01860-2.

[65] Sumon Ahmed, Magnus Rattray, and Alexis Boukouvalas. "GrandPrix: Scaling up the Bayesian GPLVM for Single-Cell Data". en. In: *bioRxiv* (Dec. 2017), p. 227843. DOI: 10.1101/227843.

[66] Jiajun Zhang and Tianshou Zhou. "Topographer Reveals Stochastic Dynamics of Cell Fate Decisions from Single-Cell RNA-Seq Data". en. In: *bioRxiv* (Jan. 2018), p. 251207. DOI: 10.1101/251207.

[67] Nan Papili Gao, Thomas Hartmann, and Rudiyanto Gunawan. "CALISTA: Clustering And Lineage Inference in Single-Cell Transcriptional Analysis". en. In: *bioRxiv* (Jan. 2018), p. 257550. DOI: 10.1101/257550.

[68] Suoqin Jin et al. "scEpath: Energy Landscape-Based Inference of Transition Probabilities and Cellular Trajectories from Single-Cell Transcriptomic Data". eng. In: *Bioinformatics (Oxford, England)* (Feb. 2018). ISSN: 1367-4811. DOI: 10.1093/bioinformatics/bty058.

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

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

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

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

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

[74] Sean Davis. *Awesome-Single-Cell: List of Software Packages for Single-Cell Data Analysis, Including RNA-Seq, ATAC-Seq, Etc*. Jan. 2018.

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

[76] Anthony Gitter. *Single-Cell-Pseudotime: An Overview of Algorithms for Estimating Pseudotime in Single-Cell RNA-Seq Data*. Jan. 2018.

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

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

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

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

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

[82] Nir Yosef et al. "Dynamic Regulatory Network Controlling Th17 Cell Differentiation". In: *Nature* 496.7446 (Apr. 2013), pp. 461–468. ISSN: 0028-0836. DOI: 10.1038/nature11981.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[98] Alex Anderson. *Writing Great Scientific Code*. http://alexanderganderson.github.io/code/2016/10/12/coding_tips.html. Oct. 2016.

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

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

[101] jfp. *Green Dice Are Loaded (Welcome to p-Hacking) (IT Best Kept Secret Is Optimization)*. en. https://www.ibm.com/developerworks/community/bl CT915. Mar. 2016.

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

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

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

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