A comparison of single-cell trajectory inference methods

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

Published in Nat Biotechnol 37, 547–554 (2019). doi: 10.1038/s41587-019-0071-9.

Abstract

Recent technological advances allow the unbiased modelling of cellular dynamic processes, by generating -omics profiles of thousands of single cells and computationally ordering these cells along a trajectory. Since 2014, at least 76 tools for trajectory inference have been developed, but due to high variability in their inputs and outputs, these methods are difficult to compare. As a result, a comprehensive assessment of the performance of trajectory inference methods is still lacking, and more importantly also guidelines for users in the field.

In this study, we evaluated a total of 47 methods in terms of cellular ordering, topology, scalability, and usability. Our results highlight the complementarity of existing tools, and that there is no generally best performing method. Instead, the choice of method mostly depends on the dataset dimensions and the trajectory topology present in the dataset. To assist users in selecting the most appropriate method, we therefore developed a set of guidelines, available as an interactive app at guidelines.dynverse.org. While the field certainly has matured significantly since its inception, further progress is still necessary in order to cope with increasingly complex and novel use cases. We hope to spearhead such developments, and the field as a whole, by making our evaluation pipeline easily extensible and freely available at github.com/dynverse/dynverse.

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 modelled computationally using trajectory inference (TI) methods, also called pseudotime analysis, which order cells along a trajectory by their overall similarity in -omics datasets.[3–5] The resulting trajectories are most often linear, bifurcating or tree-shaped, but more recent methods also allow identifying more complex trajectory topologies such as cyclic[6] or disconnected graphs.[7] TI methods offer an unbiased and transcriptome-wide understanding of a dynamic process,[1] thereby allowing the objective identification of new (primed) subsets of cells,[8] delineation of a differentiation tree[9,10] and inference of regulatory interactions responsible for one or more bifurcations.[11] Current applications of TI focus on specific subsets of cells, but ongoing efforts to construct transcriptomic catalogues of whole organisms[12–14] underline the urgency for accurate, scalable[11,15] and user-friendly TI methods.

A plethora of TI methods has been developed over the last years, and even more are being created every month (**Supplementary Table 1**). Indeed, in several repositories listing single-cell

tools, such as omictools.org,[16] the "awesome-single-cell" list (github.com/seandavi/awesome-single-cell)[17] and scRNA-tools.org,[18] TI methods are one of the largest categories. While each method has its own unique set of characteristics in terms of underlying algorithm, required prior information and produced outputs, two of the most distinctive differences between TI methods are whether they fix the topology of the trajectory, and what type(s) of graph topologies they can detect. Early TI methods typically fixed the topology algorithmically (e.g. linear[8,19–21] or bifurcating trajectories),[22,23] or through parameters provided by the user.[24,25] These methods therefore mainly focus on correctly ordering the cells along the fixed topology. Most recent methods also infer the topology,[7,26,27] which increases the difficulty of the problem at hand, but allows an unbiased identification of both the ordering inside a branch and the topology connecting these branches.

Given the diversity in TI methods, an important issue to address is a quantitative assessment of their performance, scalability, robustness and usability. Many attempts at tackling this issue have already been made,[7,22,25,28–33] but a comprehensive benchmarking evaluation of TI methods, which compares a large set of methods on a large set of datasets, is still lacking. This is problematic, as new users to the field are confronted with an overwhelming choice of TI methods, without a clear idea of which method would optimally solve their problem. Moreover, the strengths and weaknesses of existing methods need to be assessed, so that new developments in the field can focus on improving the current state-of-the-art.

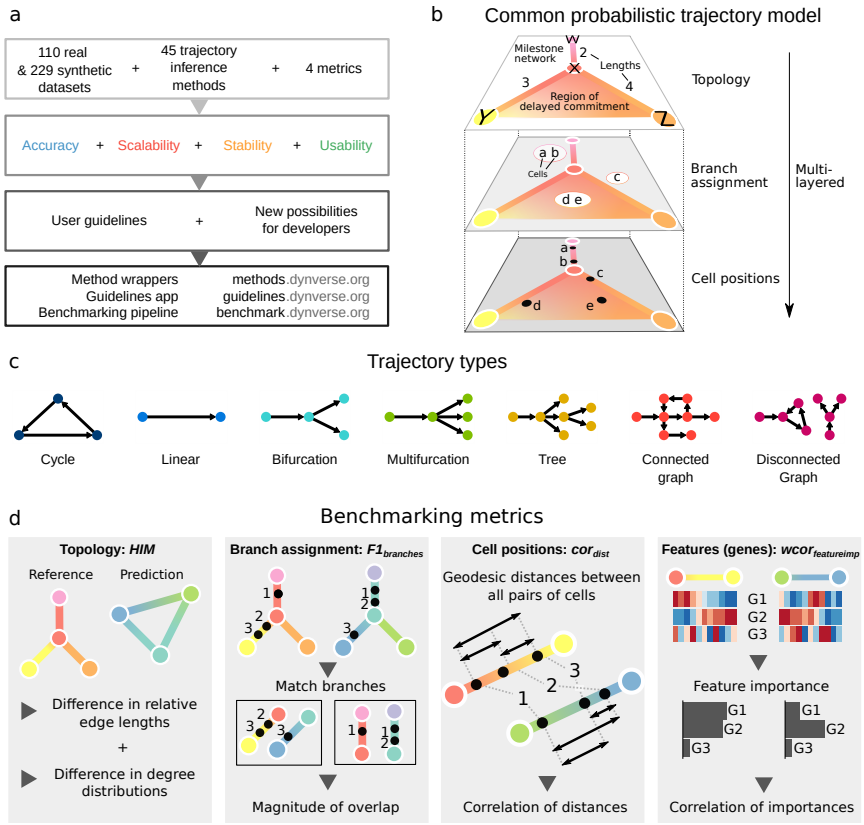

Figure 1: Overview of several key aspects of the evaluation. a) A schematic overview of our evaluation pipeline. b) In order to make the trajectories comparable to each other, a common trajectory model was used to represent reference trajectories from the real and synthetic datasets, as well as any predictions of TI methods. c) Trajectories are automatically classified into one of seven trajectory types, with increasing complexity. d) We defined four metrics, each assessing the quality of a different aspect of the trajectory. The Hamming-Ipsen-Mikhailov (HIM) score will assess the similarity between the two topologies, taking into account differences in edge lengths and degree distributions. The $F1_{branches}$ assesses the similarity of the assignment of cells onto branches. The cor_{dist} quantifies the similarity in cellular positions between two trajectories, by calculating the correlation between pairwise geodesic distances. Finally, $wcor_{features}$ quantifies the agreement between trajectory differentially expressed features from the known trajectory and the predicted trajectory.

Results

Trajectory inference methods

In this study, we performed a comprehensive evaluation of 47 TI methods (**Figure 1a**). In order to make the outputs from different methods directly comparable to each other, we developed a common probabilistic model for representing trajectories from all possible sources (**Figure 1b**). In this model, the overall topology is represented by a network of "milestones", and the cells are placed within the space formed by each set of connected milestones. Although almost every method returned a unique set of outputs, we were able to classify these outputs into 7 distinct groups (**Supplementary Figure 1**), and wrote a common output convertor for each of these groups (**Figure 2a**). When strictly required, we also provided prior information to the method. These different priors can range from weak priors which are relatively easy to acquire, such as a start cell, to strong priors, such as a known grouping of cells, which are much harder to know a priori and which can potentially introduce a large bias into the analysis (**Figure 2a**).

The largest difference between TI methods is whether a method fixes the topology, and if it does not, what kind of topologies it can detect. We defined seven possible types of topologies, ranging from very basic topologies (linear, cyclical and bifurcating) to the more complex ones (connected and disconnected graphs). Most methods either focus on inferring linear trajectories, or limit the search to tree or less complex topologies, with only a selected few attempting to infer cyclic or disconnected topologies (**Figure 2a**).

We evaluated each method on four core aspects: (i) accuracy of a prediction, given a gold or silver standard on 0 real and 0 synthetic datasets, (ii) scalability with respect to the number of cells and features (e.g. genes) (iii) stability of the predictions after subsampling the datasets, and (iv) the usability of the tool in terms of software, documentation and the manuscript. Overall, we found a large diversity across the four evaluation criteria, with only a few methods, such as PAGA, Slingshot and SCORPIUS, performing well across the board (**Figure 2b**). We will discuss each evaluation criterion in more detail (**Figure 3** and **Supplementary Figure 2**), after which we conclude with guidelines for method users and future perspectives for method developers.

Accuracy

We defined several metrics to compare a prediction to a reference trajectory (**Supplementary Note 1**). Based on an analysis of their robustness and conformity to a set of rules (**Supplementary Note 1**), we chose four metrics each assessing a different aspect of a trajectory (**Figure 1d**): the topology (Hamming-Ipsen-Mikhailov, HIM), the quality of the assignment of cells to branches ($F1_{branches}$), the cell positions (cor_{dist}) and the accuracy of the differentially expressed features along the trajectory ($wcor_{\text{features}}$). The data compendium consisted of both synthetic datasets, which offer the most exact reference trajectory, and real datasets, which provide the highest biological relevance. These real datasets come from a variety of single-cell technologies, organisms, and dynamic processes, and contain several types of trajectory topologies (**Supplementary Table 2**). Real datasets were classified as "gold standard" if the reference trajectory was not extracted from the expression data itself, such as via cellular sorting or cell mixing.[34] All other real datasets were classified as "silver standard". For synthetic datasets we used several data simulators, including a simulator of gene regulatory networks using a thermodynamic model of gene regulation[35]. For each simulation, we used a real dataset as a reference, to match its dimensions, number of differentially expressed genes, drop-out rates and other statistical properties[36].

We found that method performance was very variable across datasets, indicating that there is no "one-size-fits-all" method that works well on every dataset (**Supplementary Figure 3a**). Even methods which can detect most of the trajectory types, such as PAGA, RaceID/StemID and SLICER were not the best methods across all trajectory types (**Figure 3b**). The overall score between the different dataset sources was moderately to highly correlated (0.5 - 0.9) with the scores on real datasets containing a gold standard (**Supplementary Figure 3b**), confirming both the accuracy of the gold standard trajectories and the relevance of the synthetic data. On the other hand, the different metrics frequently disagreed with each other, with Monocle and PAGA Tree scoring better on the topology scores, while other methods, such as Slingshot, were better at ordering the cells and placing them into the correct branches (**Figure 3b**).

The performance of a method was strongly dependent on the type of trajectory present in the

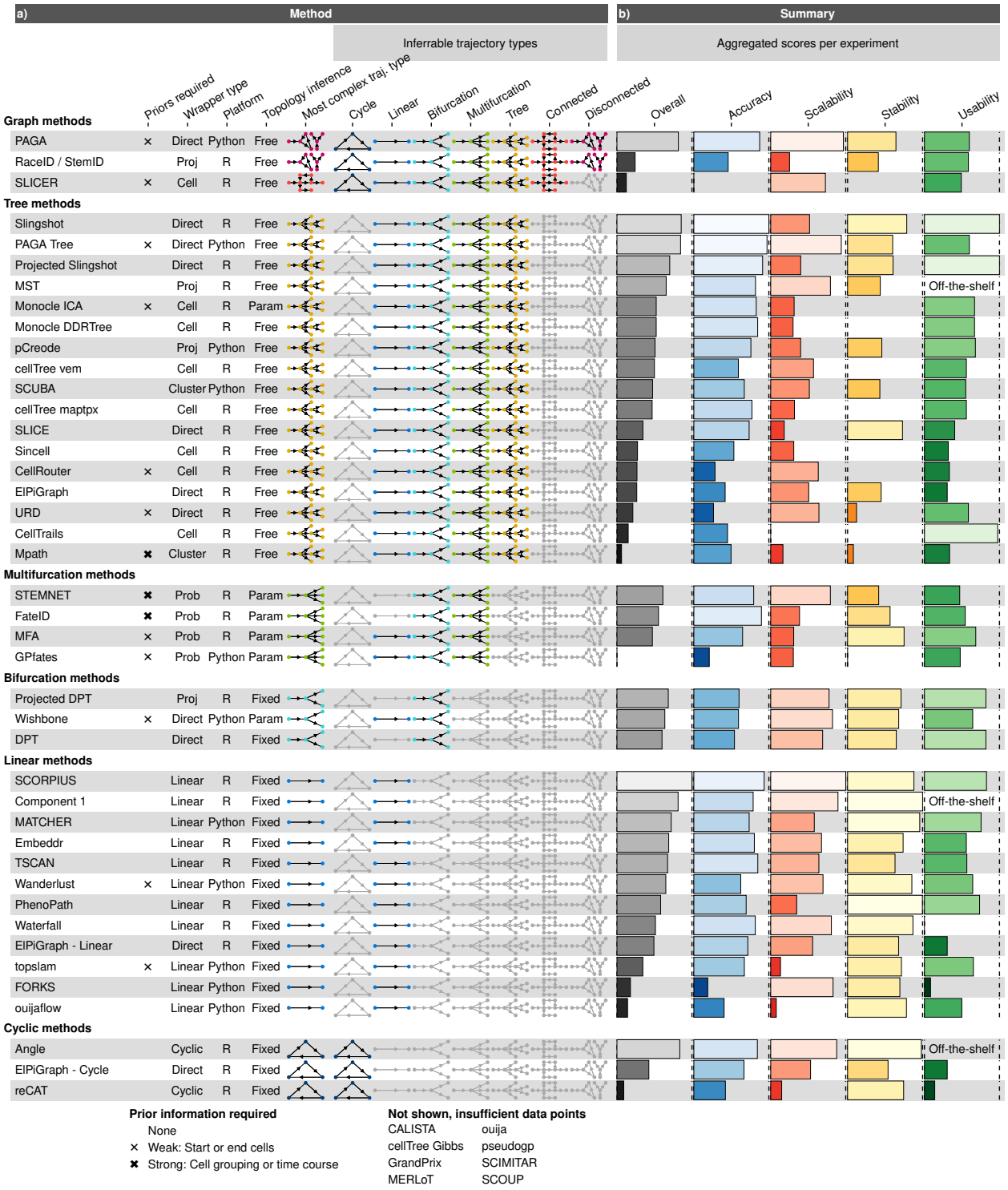

Figure 2: A characterisation of the 47 methods evaluated in this study, and their overall evaluation results. (a) We characterised the methods according to the wrapper type, their required priors, whether the inferred topology is constrained by the algorithm (fixed) or a parameter (param), and the types of inferable topologies. The methods are grouped vertically based on the most complex trajectory type they can infer. (b) The overall results of the evaluation on four criteria: accuracy using a reference trajectory on real and synthetic data, scalability with increasing number of cells and features, stability across dataset subsamples, and quality of the implementation. Methods which errored on more than 50

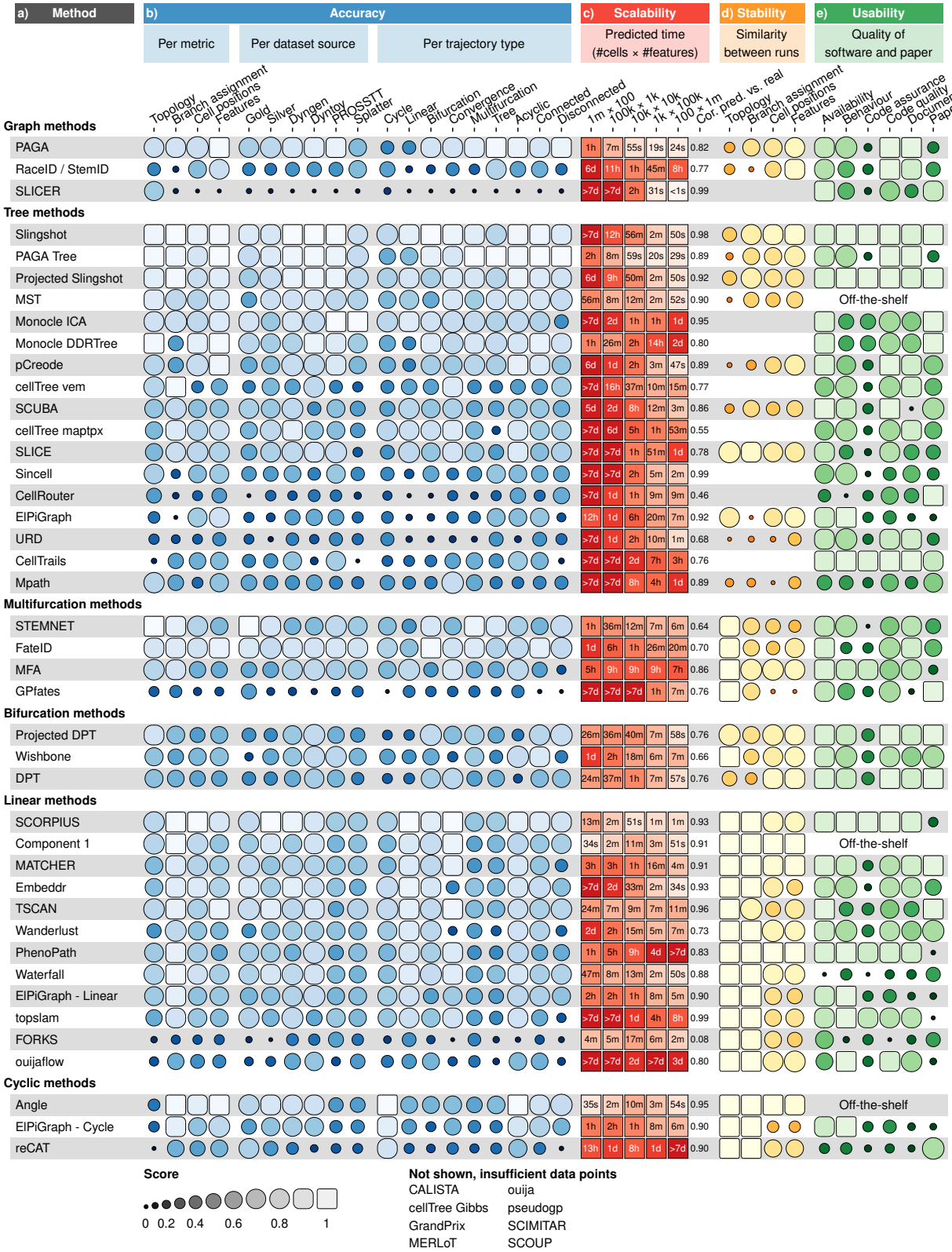

Figure 3: Detailed results of the four main evaluation criteria: accuracy, scalability, stability and usability. (a) The names of the methods, ordered as in Figure 2. (b) Accuracy of trajectory inference methods across metrics, dataset sources and dataset trajectory types. The performance of a method is generally more stable across dataset sources, but very variable depending on the metric and trajectory type. (c) Predicted execution times for varying numbers of cells and features (# cells × # features). Predictions were made by training a regression model after running each method on bootstrapped datasets with varying numbers of cells and features. (d) Stability results by calculating the average pairwise similarity between models inferred across multiple runs of the same method. (e) Usability scores of the tool and corresponding manuscript, grouped per category. Off-the-shelf methods were directly implemented in R and thus do not have a usability score.

data (**Figure 3b**). Slingshot typically performed better on datasets containing more simple topologies, while PAGA, pCreode and RaceID/StemID had higher scores on datasets with trees or more complex trajectories (**Supplementary Figure 3c**). This was reflected in the types of topologies detected by every method, as those predicted by Slingshot tended to contain less branches, while those detected by PAGA, pCreode and Monocle DDRTree gravitated towards more complex topologies (**Supplementary Figure 3d**). This analysis therefore indicates that detecting the right topology is still a difficult task for most of these methods, because methods tend to be either too optimistic or too pessimistic regarding the complexity of the topology in the data.

The high variability between datasets, together with the diversity in detected topologies between methods, could indicate some complementarity between the different methods. To test this, we calculated the likelihood of obtaining a top model when using only a subset of all methods. A top model in this case was defined as a model with an overall score of at least 95% as the best model. On all datasets, using one method would result in getting a top model in about 27% of the time. This increased to up to 74% with the addition of 6 other methods (**Figure 4a**). This resulted in a relatively diverse set of methods, containing both strictly linear or cyclic methods, and methods with a broad trajectory type range such as PAGA. We found similar indications of complementarity between the top methods on data containing only linear, bifurcation or multifurcating trajectories (**Figure 4b**), although in these cases less methods were necessary to obtain at least one top model for a given dataset. Altogether, this shows that there is considerable complementarity between the different methods, and that users should try out a diverse set of methods on their data, especially when the topology is unclear a priori. Moreover, it also opens up the possibilities for new ensemble methods which utilize this complementarity.

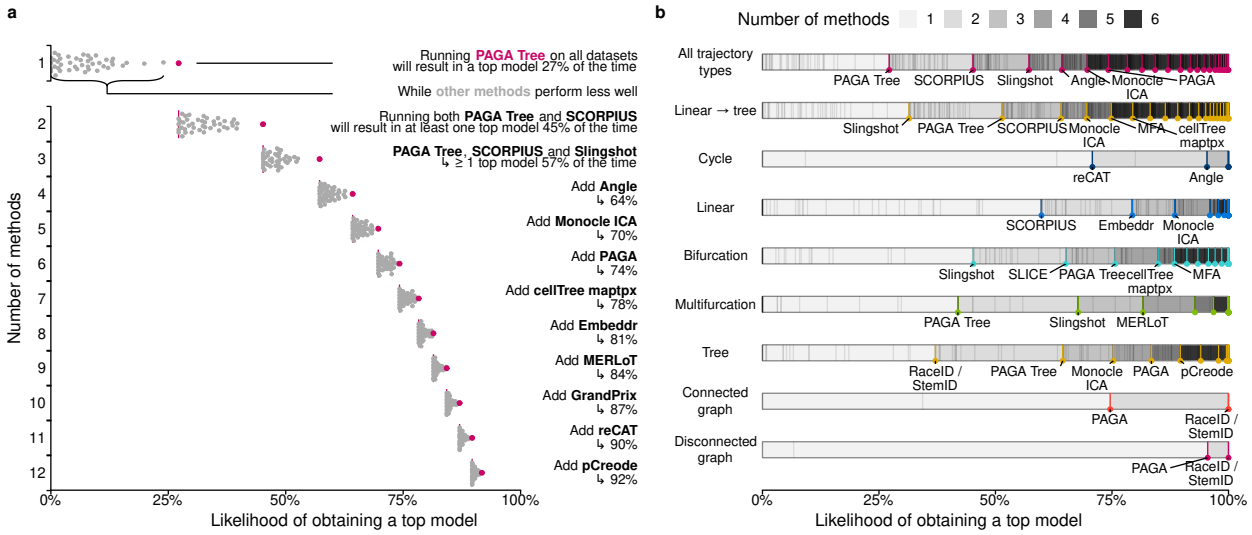

Figure 4: Complementarity between different trajectory inference methods (a) We assessed the likelihood for different combinations of methods to lead to a 'top model' (defined as a model with overall score at least 95% of the best model) when applied to all datasets. (b) The likelihood for different combinations of methods to lead to a 'top model' was assessed separately on different trajectory types. For this figure, we did not include any methods requiring strong prior information (cell grouping or time course).

Scalability

While early TI methods were developed at a time where profiling more than a thousand cells was exceptional, methods now have to cope with hundreds of thousands of cells, and perhaps soon with more than ten million.[37] Moreover, the recent application of TI methods on multi-omics single-cell data also showcases the increasing demands on the number of features.[38] To assess the scalability, we ran each method on up- and downscaled versions of five distinct real datasets. We modelled the running time and memory usage using a Shape Constrained Additive Model[39] (**Supplementary Figure 4a**). As a control, we compared the predicted time (and memory) with the actual time (resp. memory) on all benchmarking datasets, and found that these were highly correlated overall (> 0.9, **Supplementary Figure 5**), and moderately to highly correlated (0.5 - 0.9) for almost every method, depending on to what extent the execution of a method succeeded during the scalability experiments (**Figure 3c** and **Supplementary Figure 2a**).

We found that the scalability of most methods was overall very poor, with most graph and tree methods not finishing within an hour on a dataset with ten thousand cells and ten thousand features (**Figure 3c**), around the size of a typical droplet-based single-cell dataset.[37] Running times increased further with increasing number of cells, with only a handful of graph/tree methods completing within a day on a million of cells (PAGA, PAGA Tree, Monocle DDRTree, Stemnet and GrandPrix). Some methods, such as Monocle DDRTree and GrandPrix, also suffered from unsatisfactory running times when given a high number of features.

Methods with a low running time typically had two defining aspects: they had a linear time complexity with respect to the features and/or cells, and adding new cells or features led to a relatively low increase in time (**Supplementary Figure 4b**). We found that more than half of all methods had a quadratic or superquadratic complexity with respect to the number of cells, which would make it difficult to apply any of these methods in a reasonable time frame on datasets with more than a thousand cells (**Supplementary Figure 4b**).

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

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

Stability

It is not only important that a method is able to infer an accurate model in a reasonable time frame, but also that it produces a similar model when given very similar input data. In order to test the stability of each method, we executed each method on 10 different subsamples of the datasets (95% of the cells, 95% of the features), and calculated the average similarity between each pair of models using the same scores used to assess the accuracy of a trajectory (**Figure 3d**).

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

Usability

While not directly related to the accuracy of the inferred trajectory, it is also important to assess the quality of the implementation and how easy it is for a biological user to use.[40] We scored each method using a transparent checklist of important scientific and software development practices, including software packaging, documentation, automated code testing, and publication into a peer-reviewed journal (**Supplementary Table 3**). Important to note here is that there is a selection bias in the tools selected for this analysis, as we did not include a substantial set of tools due to issues with installation, code availability and executability on a freely available platform (which excludes MATLAB). The reason for not including certain tools are all discussed on our repository (github.com/dynverse/dynmethods/issues?q=label:"won't+wrap"). Installation issues seem to be quite general in bioinformatics,[41] and the trajectory inference field is no exception.

We found that most methods fulfilled the basic criteria, such as the availability of a tutorial and elemental code quality criteria (**Figure 3d** and **Supplementary Figure 6**). While recent methods had a slightly better quality score than older methods, several quality aspects were consistently lacking for the majority of the methods (**Supplementary Figure 6 right**) and we believe that these should receive extra attention from developers. Although these outstanding issues covered all five categories, code assurance and documentation in particular were problematic areas, notwithstanding several studies pinpointing these as good practices.[42,43] Only two methods had

a nearly perfect usability score (Slingshot and Celltrails), and these could be used as an inspiration of future methods. We observed no clear relation between method quality and method performance (**Figure 2b**).

Discussion

In this study, we presented a large scale evaluation of the performance of 47 TI methods. By using a common trajectory representation and four metrics to compare the methods' outputs, we were able to assess the accuracy of the methods on more than two-hundred datasets. We also assessed several other important quality measures, such as the quality of the method's implementation, the scalability to hundreds of thousands of cells, and the stability of the output on small variations of the datasets.

Based on the results of our benchmark, we propose a set of practical guidelines for method users (**Figure 5** and guidelines.dynverse.org). We postulate that, as a method's performance is heavily dependent on the trajectory type being studied, the choice of method should currently be primarily driven by the anticipated trajectory topology in the data. For 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 disconnected trajectories, cycles or a complex tree structure. In each of these use cases, our evaluation suggests a different set of optimal methods, as shown in **Figure 5**. Several other factors will also impact the choice of methods, such as the dimensions of the dataset, and the prior information which is available. These factors and several others can all be dynamically explored in our interactive app (guidelines.dynverse.org). This app can also be used to query the results of this evaluation, such as filtering the datasets, or changing the importance of the evaluation metrics for the final ranking.

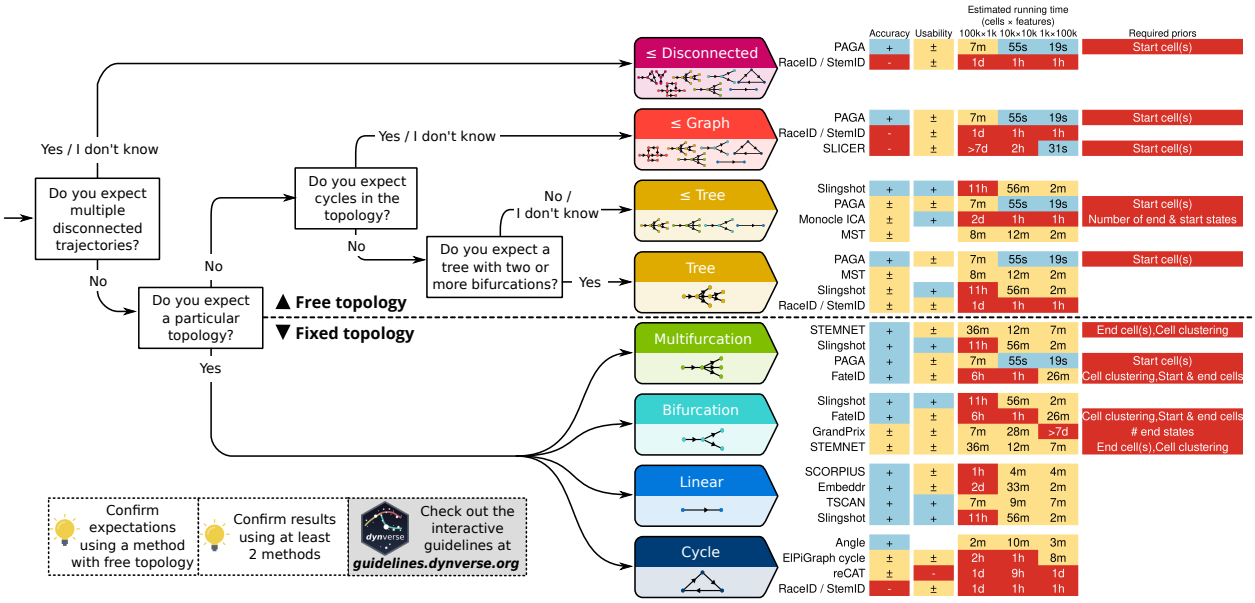

Figure 5: Practical guidelines for method users. As the performance of a method mostly depends on the topology of the trajectory, the choice of TI method will be primarily influenced by the user's existing knowledge about the expected topology in the data. We therefore devised a set of practical guidelines, which combines the method's performance, user friendliness and the number of assumptions a user is willing to make about the topology of the trajectory. Methods to the right are ranked according to their performance on a particular (set of) trajectory type. Further to the right are shown the accuracy (+: scaled performance ≥ 0.9, ±: > 0.6), usability scores (+: ≥ 0.9, ± ≥ 0.6), estimated running times and required prior information.

When inferring a trajectory on a data set of interest, it is important to take two further points into account. First, it is critical that a trajectory, and the downstream results and/or hypotheses originating from it, are confirmed by multiple TI methods. This is to make sure the prediction is not biased due to the given parameter setting or the particular algorithms underlying a TI method. The value of using different methods is further supported by our analysis indicating substantial

complementarity between the different methods. Second, even if the expected topology is known, it can be beneficial to also try out methods which make less assumptions about the trajectory topology. When the expected topology is confirmed using such a method, it provides additional evidence to the user. When a more complex topology is produced, this could indicate that the underlying biology is much more complex than anticipated by the user.

Critical to the broad applicability of TI methods is the standardisation of the input and output interfaces of TI methods, so that users can effortlessly execute TI methods on their dataset of interest, compare different predicted trajectories, and apply downstream analyses such as finding genes important for the trajectory, network inference[11] or finding modules of genes.[44] Our framework is an initial attempt at tackling this problem, and we illustrate its usefulness here by comparing the predicted trajectories of several top-performing methods on datasets containing a linear, tree, cyclic and disconnected graph topology (**Figure 6**). Using our framework, this figure can be recreated using only a couple of lines of R code (methods.dynverse.org). In the future, this framework could be extended to allow additional input data, such as spatial and RNA velocity information,[45] and easier downstream analyses. In addition, further discussion within the field is required in order to arrive at a consensus concerning a common interface for trajectory models, which can include additional features such as uncertainty and gene importance.

Our study indicates that the field of trajectory inference is maturing, primarily for linear and bifurcating trajectories (as we illustrate with two datasets in **Figure 6a and b**). However, we also highlight several ongoing challenges, which should be addressed before TI can be a reliable tool for analysing single-cell -omics datasets with complex trajectories. Foremost, new methods should focus on improving the unbiased inference of tree, cyclic graph and disconnected topologies, as we found that methods repeatedly overestimate or underestimate the complexity of the underlying topology, even if the trajectory could easily be identified using a dimensionality reduction method (illustrated with some synthetic datasets in **Figure 6c and d**). Furthermore, higher standards for code assurance and documentation could help in adopting these tools across the single-cell -omics field. Finally, new tools should be designed to scale well with the increasing number of cells and features. We found that only a handful of current methods are able to handle datasets with more than 10.000 cells within a reasonable timeframe, making the others possibly obsolete very soon given the rapid technological developments. To support the development of these new tools, we provide a series of vignettes on how to wrap and evaluate a method and on the different measures proposed in this study at benchmark.dynverse.org.

We found that the performance of a method can be very variable between datasets, and therefore included a large set of both real and synthetic data within our evaluation, leading to a robust overall ranking of the different methods. Nonetheless, we want to avoid that a race to be the first on the ranking would stifle creativity, given that there is a high risk that a certain new methodology would not improve upon the state-of-the-art. Indeed, we firmly believe that "good-yet-not-the-best" methods[46] can still provide a very valuable contribution to the field, especially if they make use of novel algorithms, return a more scalable solution, or provide a unique insight in specific use cases. This is also supported by our analysis of method complementarity. Some examples for the latter include PhenoPath, which can include additional covariates in its model, Ouija, which returns a measure of uncertainty of each cell's position within the trajectory, and StemID, which can infer the directionality of edges within the trajectory. We feel that such methods should also be valued, and encourage their use provided that they are contained in a user friendly tool.

Online Methods

Data and code availability

The processed real and synthetic datasets used in this study are deposited on Zenodo (doi.org/10.5281/zenodo.1443566).[47]

The main analysis repository is available at benchmark.dynverse.org and is divided into several experiments. Each experiment has its own set of scripts and results, each accompanied by an illustrated readme which can be easily browsed and explored on the github website.

The analysis scripts call several other R packages, of which an overview is available at dynverse.org. These packages include **dynwrap**, used to wrap the output of methods into the

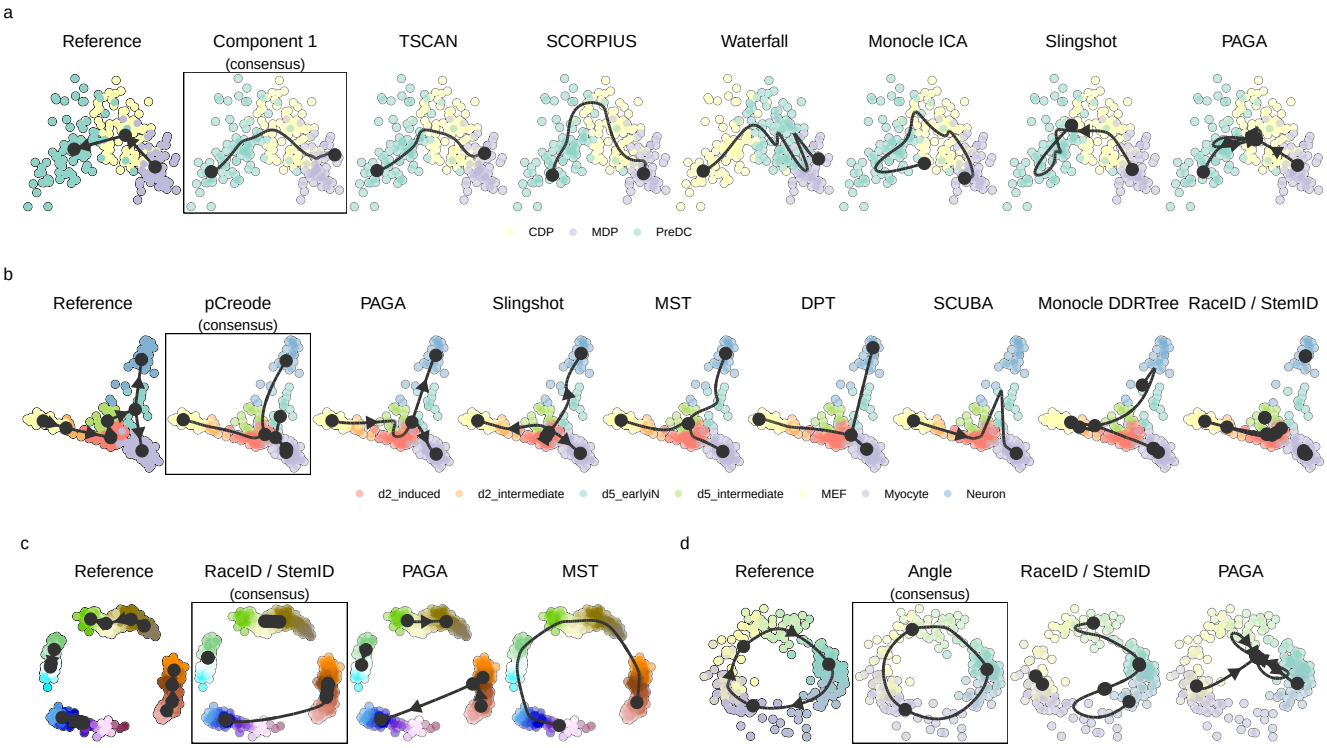

Figure 6: Demonstration of how a common framework for TI methods facilitates broad applicability using some example datasets. Trajectories inferred by each method were projected to a common dimensionality reduction using multi-dimensional scaling. For each dataset, we also calculated a "consensus" prediction, by calculating the cor_{dist} between each pair of models, and picking the model with the highest score on average. (a) The top methods applied on a dataset containing a linear trajectory of differentiation dendritic cells, going from MDP, CDP to PreDC. (b) The top methods applied on a dataset containing a bifurcating trajectory of reprogrammed fibroblasts. (c) A synthetic dataset generated by dyntoy, containing four disconnected trajectories. (d) A synthetic dataset generated by dyngen, containing a cyclic trajectory.

common trajectory model, **dyneval**, which contains the evaluation metrics, **dynguidelines**, the guidelines app, and **dynplot** for plotting trajectories.

Trajectory inference methods

We gathered a list of 76 trajectory inference tools (**Supplementary Table 1**), by searching the literature for "trajectory inference" and "pseudotemporal ordering", and based on two existing lists found online: github.com/seandavi/awesome-single-cell[17] and github.com/agitter/single-cell-pseudotime.[48] We welcome any contributions by creating an issue at methods.dynverse.org.

Methods were excluded from the evaluation based on several criteria: (a) Not freely available, (b) No code available, (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). The discussions on why these methods were excluded can be found at github.com/dynverse/dynmethods/issues?q=label:"won't wrap". In the end, we included 47 methods in the evaluation.

Method wrappers

To make easy to run each method in a reproducible manner, each method was wrapped within docker and singularity containers (available at methods.dynverse.org). These containers are automatically built on both Singularity Hub (singularity-hub.org) and Docker Hub (hub.docker.com/u/dynverse). For each method, we wrote a wrapper script based on example scripts or tutorials provided by the authors (as mentioned in the respective wrapper scripts). This script reads in the input data, runs the method, and outputs the files required to construct a trajectory. We also created a script to generate an example dataset, which is used to automatically test every method with continuous integration on travis-ci.org (travis-ci.org/dynverse).

We used the github issues system to contact the authors of each method, and asked for feedback on the wrappers, the metadata, and the quality control. About one third of the authors responded, and we improved the wrappers based on their feedback. These discussions can be viewed on github: github.com/dynverse/dynmethods/issues?q=label:"method+discussion".

Method input

As input, we provided each method with either the raw count data (after cell and gene filtering) or normalised expression values, based on the description in the method documentation or from the study describing the method. A large portion of the methods requires some form of prior information (e.g. a start cell) in order to be executable. Other methods optionally allow to exploit certain prior information. Prior information can be supplied as a starting cell from which the trajectory will originate, a set of important marker genes, or even a grouping of cells into cell states. Providing prior information to a TI method can be both a blessing and a curse. In one way, prior information can help the method to find the correct trajectory among many, equally likely, alternatives. On the other hand, incorrect or noisy prior information can bias the trajectory towards current knowledge. Moreover, prior information is not always easily available, and its subjectivity can therefore lead to multiple equally plausible solutions, restricting the applicability of such TI methods to well studied systems.

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

- **Start cells** The identity of one or more start cells. For both real and synthetic data, a cell was chosen which was the closest (in geodesic distance) to each milestone with only outgoing edges. For ties, one random cell was chosen. For cyclic datasets, a random cell was chosen.
- **End cells** The identity of one or more end cells. Similar as the start cells, but now for every state with only incoming edges.
- **# end states** Number of terminal states. Number of milestones with only incoming 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.
- **Discrete time course** For each cell a time point from which it was sampled. If available, directly extracted from the reference trajectory, otherwise the geodesic distance from the root milestone was used. For synthetic data, four discrete timepoints were chosen, at which the cells were "sampled" to provide a time course information reflecting the one provided in real experiments.
- **Continuous time course** For each cell a time point from which it was sampled. For real data this was equal to the discrete time course, for synthetic data we used the internal simulation time of each simulator.

Common trajectory model

Due to the absence of a common format for trajectory models, most methods return a unique set of output formats with few overlaps. We therefore post-processed the output of each method into a common probabilistic trajectory model (**Supplementary Figure 1a**). This model consisted 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 regions of delayed commitment, which defines connections between three or more milestones. These must be explicitly defined in the trajectory model and per region one milestone must be directly connected to all other milestones of the region.

Depending on the output of a method, we used different strategies to convert the output to our model (**Supplementary Figure 1b**). Special conversions are denoted by an *, and will be explained in more detail below.

- **Type 1, direct:** CALISTA*, DPT*, ElPiGraph, ElPiGraph - Cycle, ElPiGraph - Linear, MERLoT, PAGA, PAGA Tree, Projected Slingshot, SLICE*, Slingshot, URD* and Wishbone. The wrapped method directly returned a network of milestones, the regions of delayed commitment, and for each cell it is given to what extent it belongs to a milestone. In some cases, this indicates that additional transformations were required for the method, not covered by any of the following output formats. Some methods returned a branch network instead of a milestone network, and this network was converted by calculating the line graph of the branch network.
- **Type 2, linear pseudotime:** Component 1, Embeddr, FORKS, MATCHER, ouija, ouijaflow, PhenoPath, pseudogp, SCIMITAR, SCORPIUS, topslam, TSCAN, Wanderlust and Waterfall. The method returned a pseudotime, which is translated into a linear trajectory where the milestone network contains two milestones and cells are positioned between these two milestones.
- **Type 3, cyclical pseudotime:** Angle, Oscope and reCAT. The method returned a pseudotime, which is translated into a cyclical trajectory where the milestone network contains three milestones and cells are positioned between these three milestones. These milestones were positioned at pseudotime 0, ⅓ and ⅔ .
- **Type 4, end state probability:** FateID, GPfates, GrandPrix, MFA*, SCOUP and STEMNET. The method returned a pseudotime and for each cell and end state a probability (Pr) for how likely a cell will end up in a certain end state. This was translated into a star-shaped milestone network, with one starting milestone (M_0) and several outer milestones (M_i), with regions of delayed commitment between all milestones. The milestone percentage of a cell to one of the outer milestones was equal to pseudotime $\times$ Pr_{M_i}. The milestone percentage to the starting milestone was equal to $1 -$ pseudotime .
- **Type 5, cluster assignment:** Mpath and SCUBA. The method returned a milestone network, and an assignment of each cell to a specific milestone. Cells were positioned onto the milestones they are assigned to, with milestone percentage equal to 1.
- **Type 6, orthogonal projection:** MST, pCreode, Projected DPT and RaceID / StemID. The method returned a milestone network, and a dimensionality reduction of the cells and milestones. The cells were projected to the closest nearest segment, thus determining the cells' position along the milestone network. If a method also returned a cluster assignment (type 5), we limited the projection of each cell to the closest edge connecting to the milestone of a cell. For these methods, we usually wrote two wrappers, one which included the projection, and one without.

- **Type 7, cell graph:** CellRouter, CellTrails, cellTree Gibbs, cellTree maptpx, cellTree vem, Monocle DDRTree, Monocle ICA, Sincell* and SLICER. The method returned a network of cells, and which cell-cell transitions are part of the 'backbone' structure. Backbone cells with degree $\neq 2$ were regarded as milestones, and all other cells were placed on transitions between the milestones. If methods did not return a distance between pairs of cells, the cells were uniformly positioned between the two milestones. Otherwise, we first calculated the distance between two milestones as the sum of the distances between the cells, and then divided the distance of each pair of cells with the total distance to get the milestone percentages.

Special conversions were necessary for certain methods:

- **CALISTA:** We assigned the cells to the branch at which the sum of the cluster probabilities of two connected milestones was the highest. The cluster probabilities of the two selected milestones were then used as milestone percentages. This was then processed as a type 1, direct, method.
- **DPT:** We projected the cells onto the cluster network, consisting of a central milestone (this cluster 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 a type 1, direct, method.
- **Sincell:** To constrain the number of milestones this method creates, we merged two cell clusters iteratively until the percentage of leaf nodes was below a certain cutoff, default at 25%. This was then processed as a type 7, cell graph, method.
- **SLICE:** As discussed in the vignette of SLICE, we ran principal curves one by one for every edge detected by SLICE. This was then processed as a type 1, direct, method.
- **MFA:** We used the branch assignment as state probabilities, which together with the global pseudotime were processed as a type 4, end state probabilities, method.
- **URD:** We extracted the pseudotime of a cell within each branch using the y positions in the tree layout. This was then further processed as a type 1, direct, method.

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

Off-the-shelf methods

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

- **Component 1:** This method returns the first component of a PCA dimensionality reduction as a linear trajectory. This method is especially relevant as it has been used in a few studies already.[49,50]
- **Angle:** Similar to the previous method, this method computes the angle with respect to the origin in a two dimensional PCA, and uses this angle as a pseudotime for generating a cyclical trajectory.
- **MST:** This method performs PCA dimensionality reduction, followed by clustering using the R mclust package, after which the clusters are connected using a minimum spanning tree. The trees are orthogonally projected to the nearest segment of the tree. This baseline is highly relevant as many methods follow the same methodology: dimensionality reduction, clustering, topology inference, project cells to topology.

Trajectory types

We classified all possible trajectory topologies into distinct trajectory types, based on topological criteria (**Figure 1c**). These trajectory types start from the most general trajectory type, a disconnected graph, and move down (within a directed acyclic graph structure), progressively becoming more simple until the two basic types: linear and cyclical. A disconnected graph is a graph in which only one edge can exist between two nodes. A (connected) graph is a disconnected graph in which all nodes are connected. An acyclic graph is a graph containing no cycles. A tree is an acyclic graph containing no convergences (no nodes with in-degree higher than 1). A binary tree is a rooted tree with every node having maximally two outgoing edges. A convergence is a

directed acyclic graph in which only one node has a degree larger than one and this same node has an indegree of one. A multifurcation is a rooted tree in which only one node has a degree larger than one. A bifurcation is a multifurcation in which only one node has a degree equal to 3. A linear topology is a graph in which no node has a degree larger than 3. Finally, a cycle is a connected graph in which no node has a degree larger than 3 but contains one cycle. In most cases, a method which was able to detect a complex trajectory type, was also able to detect less complex trajectory types, with some exceptions shown in **Figure 2a**.

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

Real datasets

We gathered real datasets by searching for "single-cell" at the Gene Expression Omnibus and selecting those datasets in which the cells are sampled from different stages in a dynamic process (**Supplementary Table 2**). The scripts to download and process these datasets are available on our repository (benchmark.dynverse.org/tree/master/scripts/01-datasets). Whenever possible, we preferred to start from the raw counts data. These raw counts were all normalised and filtered using a common pipeline, as discussed later. Some original datasets contained more than one trajectory, in which case we split the dataset into its separate connected trajectory, but also generated several combinations of connected trajectories to include some datasets with disconnected trajectories in the evaluation. In the end, we included 0 datasets for this evaluation.

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

Synthetic datasets

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

- **dyngen:** Simulations of gene regulatory networks, available at github.com/dynverse/dyngen
- **dyntoy:** Random gradients of expression in the reduced space, available at github.com/dynverse/dyntoy
- **PROSSTT:** Expression is sampled from a linear model which depends on pseudotime[51]
- **Splatter:** Simulations of non-linear paths between different expression states[36]

For every simulator, we took great care to make the datasets as realistic as possible. To do this, we extracted several parameters from all real datasets. We calculated the number of differentially expressed features within a trajectory using a two-way Mann–Whitney U test between every pair of cell groups. These values were corrected for multiple testing using the Benjamini-Hochberg procedure (FDR < 0.05) and we required that a gene was expressed in at least 5% of cells, and had at least a fold-change of 2. We also calculated several other parameters, such as dropout rates and library sizes using the Splatter package.[36] These parameters were then given to the simulators when applicable, as described for each simulator below. Not every real dataset was selected to serve as a reference for a synthetic dataset. Instead, we chose a set of ten distinct reference real datasets by clustering all the parameters of each real dataset, and used the reference real datasets at the cluster centers from a pam clustering (with $k = 10$, implemented in the R cluster package) to generate synthetic data.

dyngen

The dyngen (github.com/dynverse/dyngen) workflow to generate synthetic data is based on the well established workflow used in the evaluation of network inference methods[35,52] and consists of four main steps: network generation, simulation, gold standard extraction and simulation of the

scRNA-seq experiment. At every step, we tried to mirror real regulatory networks, while keeping the model simple and easily extendable. We simulated a total of 110 datasets, with 11 different topologies.

Network generation One of the main processes involved in cellular dynamic processes is gene regulation, where regulatory cascades and feedback loops lead to progressive changes in expression and decision making. The exact way a cell chooses a certain path during its differentiation is still an active research field, although certain models have already emerged and been tested in vivo. One driver of bifurcation seems to be mutual antagonism, where genes[53] strongly repress each other, forcing one of the two to become inactive.[54] Such mutual antagonism can be modelled and simulated.[55,56] Although such a two-gene model is simple and elegant, the reality is frequently more complex, with multiple genes (grouped into modules) repressing each other.[57]

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

From these module networks we generated gene regulatory networks in two steps: the main regulatory network was first generated, and extra target genes from real regulatory networks were added. For each dataset, we used the same number of genes as were differentially expressed in the real datasets. 5% of the genes were assigned to be part of the main regulatory network, and were randomly distributed among all modules (with at least one gene per module). We sampled edges between these individual genes (according to the module network) using a uniform distribution between 1 and the number of possible targets in each module. To add additional target genes to the network, we assigned every regulator from the network to a real regulator in a real network (from regulatory circuits),[58] 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.[52] In this model, the changes in gene expression (x_i) and protein expression (y_i) are modeled using ordinary differential equations[35] (ODEs):

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

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

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

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

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

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

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

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

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

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

P_i is only proportional to ν because ν is 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,[35] 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.[35] These SDEs were simulated using the Euler–Maruyama approximation, with time-step $h = 0.01$ and noise strength $\eta = 8$. The total simulation time varied between 5 for linear and bifurcating datasets, 10 for consecutive bifurcating, trifurcating and converging datasets, 15 for bifurcating converging datasets and 30 for linear long, cycle and bifurcating loop datasets. The burn-in period was for each simulation 2. Each network was simulated 32 times.

Simulation of the single-cell RNA-seq experiment For each dataset we sampled the same number of cells as were present in the reference real dataset, limited to the simulation steps after burn-in. These cells were sampled uniformly across the different steps of the 32 simulations. Next, we used the Splatter package[36] to estimate the different characteristics of a real dataset, such as the distributions of average gene expression, library sizes and dropout probabilities. We used Splatter to simulate the expression levels $\lambda_{i,j}$ of housekeeping genes i (to match the number of genes in the reference dataset) in every cell j. These were combined with the expression levels of the genes simulated within a trajectory. Next, true counts were simulated using $Y'_{i,j} \sim \text{Poisson}(\lambda_{i,j})$. Finally, we simulated dropouts by setting true counts to zero by sampling from a Bernoulli distribution using a dropout probability $\pi^D_{i,j} = \frac{1}{1+e^{-k(\ln(\lambda_{i,j})-x_0)}}$. Both x_0 (the midpoint for the dropout logistic function) and k (the shape of the dropout logistic function) were estimated by Splatter.

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

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

each branch within the trajectory. To do this, we first defined in which order the expression of the modules is expected to change, and then generated a backbone expression profile in which the expression of these modules increases and decreases smoothly between 0 and 1. We also smoothed the expression in each simulation using a rolling mean with a window of 50 time steps, and then calculated the average module expression along the simulation. We used dynamic time warping, implemented in the dtw R package,[59,60] with an open end to align a simulation to all possible module progressions, and then picked the alignment which minimised the normalised distance between the simulation and the backbone. In case of cyclical trajectory topologies, the number of possible milestones a backbone could progress through was limited to 20.

dyntoy

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

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

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

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

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

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

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

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

PROSSTT

PROSSTT is a recent data simulator,[51] which simulates expression using linear mixtures of expression programs and random walks through the trajectory. We used 5 topology generators from dyntoy (linear, bifurcating, multifurcating, binary tree and tree), and simulated for each topology

generator 10 datasets using different reference real datasets. However, due to frequent crashes of the tool, only 19 datasets created output and were thus used in the evaluation.

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

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

Splatter

Splatter[36] simulates expression values by constructing non-linear paths between different states, each having a distinct expression profile. We used 5 topology generators from dyntoy (linear, bifurcating, multifurcating, binary tree and tree), and simulated for each topology generator 10 datasets using different reference real datasets, leading to a total of 50 datasets.

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

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

Dataset filtering and normalisation

We used a standard single-cell RNA-seq preprocessing pipeline which applies parts of the scran and scater Bioconductor packages.[61] 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 (which we set to 3). We required that a gene was expressed in at least 5% of the cells, and that it should have an average expression higher than 0.02. Furthermore, we used the pipeline to select the most highly variable genes, using a false discovery rate of 5% and a biological component higher than 0.5. As a final filter, we removed both all-zero genes and cells until convergence.

Benchmark metrics

The importance of using multiple metrics to compare complex models has been stated repeatedly.[46] Furthermore, a trajectory is a model with multiple layers of complexity, which calls for several metrics each assessing a different layer. We therefore defined several possible metrics for comparing trajectories, each investigating different layers. These are all discussed in **Supplementary Note 1** along with examples and robustness analyses when appropriate.

Next, we created a set of rules to which we think a good trajectory metric should conform, and tested this empirically for each metric by comparing scores before and after perturbing a dataset (**Supplementary Note 1**). Based on this analysis, we chose 4 metrics for the evaluation, each assessing a different aspect of the trajectory: (i) the HIM measures the topological similarity, (ii) the $F1_{branches}$ compares the branch assignment, (iii) the cor_{dist} assesses the similarity in pairwise cell-cell distances and thus the cellular positions, and (iv) the $wcor_{\text{features}}$ looks at whether similar important features (genes) are found in both the reference dataset and the prediction.

The Hamming-Ipsen-Mikhailov metric

The Hamming-Ipsen-Mikhailov (HIM) metric[62] uses the two weighted adjacency matrices of the milestone networks as input (weighted by edge length). It is a linear combination of the normalised Hamming distance, which gives an indication of the differences in edge lengths, and the normalised Ipsen-Mikhailov distance, which assesses the similarity in degree distributions. The

latter has a parameter γ, which was fixed at 0.1 as to make the scores comparable between datasets. We illustrate the metric and discuss alternatives in **Supplementary Note 1**.

The F1 between branch assignments

To compare branch assignment, we used an F_1 score, also used used for comparing biclustering methods.[44] To calculate this metric, we first calculated the similarity of all pairs of branches between the two trajectories using the Jaccard similarity. Next, we define the *Recovery* (respectively *Relevance*) as the average maximal similarity of all branches in the reference dataset (respectively *prediction*). The $F1_{branches}$ was then defined as the harmonic mean between *Recovery* and *Relevance*. We illustrate this metric further in **Supplementary Note 1**.

Correlation between geodesic distances

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

The correlation between important features

The $wcor_{\text{features}}$ assesses whether the same trajectory differentially expressed features are found between the prediction and the model. To calculate this metric, we used Random Forest regression (implemented in the R ranger package),[63] to predict expression values of each gene, based on the geodesic distances of a cell to each milestone. We then extract feature importance values for each feature, and calculated the similarity of the feature importances using a weighted Pearson correlation, weighted by the feature importance in the reference dataset as to give more weight to large differences. As hyperparameters we set the number of trees to 10000, and the number of features on which to split to 1% of all available features. We illustrate this metric and assess the effect of its hyperparameters in **Supplementary Note 1**.

Score aggregation

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

To make sure easy and difficult datasets have equal influence on the final score, we first normalised the scores on each dataset across the different methods. We shifted and scaled the scores to $\sigma = 1$ and $\mu = 0$, and then applied the unit probability density function of a normal distribution on these values to get the scores back into the $[0, 1]$ range.

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

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

Method execution

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

Complementarity

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

Scalability

To assess the scalability of each method, we started from 5 real datasets, selected using the centers from a k-medoids as discussed in the synthetic data generation section. We up- and downscaled these datasets between 10 to 100.000 cells and 10 to 100.000 features, while never going higher than 1.000.000 values in total. To generate new cells or features, we first generated a 10-nearest neighbour graph of both the cells and features from the expression space. For every new cell or feature, we used a linear combination of one to three existing cells or features, where each cell or features was given a weight sampled from $U(0, 1)$.

We ran each method on each dataset for maximally 1 hour and gave each process 10GB of memory. To determine the running time of each method, we started the timer right after data loading and the loading of any packages, and stopped the clock before post-processing and saving of the output. Pre- and postprocessing steps specific to a method, such as dimensionality reduction and gene filtering, were included in the time. To estimate the maximal memory usage, we used the `max_vmem` value from the `qacct` command provided by a gridengine cluster. We acknowledge however that these memory estimates are very noisy, and the averages provided in this study are therefore only rough estimates.

The relationship between the dimensions of a dataset and the running time or maximal memory usage was modelled using shape constrained additive models,[39] with $log_{10}|\textit{cells}|$ and $log_{10}|\textit{features}|$ as predictor variables, and fitted this model using the `scam` function as implemented in the R scam package, with $log_{10}\textit{time}$ (or $log_{10}\textit{memory}$) as outcome.

To classify the time complexity of each method with respect to the number of cells, we predicted the running time at 10.000 features with increasing number of cells from 100 to 100.000, with steps of 100. We trained a generalised linear model with the following function: `y ~ log(x) + sqrt(x) + x + I(x^2) + I(x^3)`. The time complexity of a method was then classified as follows:

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

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

Stability

In the ideal case, a method should produce a similar trajectory, even when the input data is slightly different. However, running the method multiple times on the same input data would not

be the ideal approach to assess its stability, given that a lot of tools are artificially deterministic by internally resetting the pseudorandom number generator (for example, using the `set.seed` function in R or the `random.seed` function in numpy). To assess the stability of each method, we therefore selected a number of datasets, which consisted of 25% of the datasets accounting for 15% of the total runtime, chosen such that after aggregation the overall scores still has > 0.99 correlation with the original overall ranking. We subsampled each dataset 10 times with 95% of the original cells and 95% of the original features. We ran every method on each of the bootstraps, and assessed the stability by calculating the benchmarking scores between each pair of subsequent models (run i is compared to run $i+1$). For the cor_{dist} and $F1_{branches}$, we only used the intersection between the cells of two datasets, while the intersection of the features was used for the $wcor_{\text{features}}$.

Usability

We created a transparent scoring scheme to quantify the usability of each method based on several existing tool quality and programming guidelines in literature and online (**Supplementary Table 3**). The goal of this quality control is in the first place 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[64] 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 datasets in which the usefulness of the method was shown, and the scope of method evaluation in the paper.

Each quality aspect received a weight depending on how frequently it was found in several papers and online sources which discuss tool quality (**Supplementary Table 3**). This was to make sure that more important aspects, such as the open source availability of the method, outweighed other less important aspects, such as the availability of a graphical user interface. For each aspect, we also assigned a weight to the individual questions being investigated (**Supplementary Table 3**). For calculating the final score, we weighed each of the six categories equally.

Guidelines

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

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

Acknowledgements

We would like to thank the original authors of the methods for their feedback and improvements on the method wrappers. This study was supported by the Fonds Wetenschappelijk Onderzoek (FWO, R.C. and W.S.) and BOF (Ghent University, H.T).

Author contributions

R.C., W.S., H.T. and Y.S. designed the study. R.C. and W.S. performed the experiments and analysed the data. W.S., R.C., and H.T. implemented software packages. R.C., W.S., Y.S. and H.T. prepared the manuscript. Y.S. supervised the project.

Competing financial interests

The authors declare no competing financial interests

Supplementary References

1. Tanay, A. & Regev, A. Scaling single-cell genomics from phenomenology to mechanism. *Nature* **541**, nature21350 (2017).

2. Etzrodt, M., Endele, M. & Schroeder, T. Quantitative Single-Cell Approaches to Stem Cell Research. *Cell Stem Cell* **15**, 546–558 (2014).

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

4. Cannoodt, R., Saelens, W. & Saeys, Y. Computational methods for trajectory inference from single-cell transcriptomics. *European Journal of Immunology* **46**, 2496–2506 (2016).

5. Moon, K. R. *et al.* Manifold learning-based methods for analyzing single-cell RNA-sequencing data. *Current Opinion in Systems Biology* **7**, 36–46 (2018).

6. Liu, Z. *et al.* Reconstructing cell cycle pseudo time-series via single-cell transcriptome data. *Nature Communications* **8**, 22 (2017).

7. Wolf, F. A. *et al.* Graph abstraction reconciles clustering with trajectory inference through a topology preserving map of single cells. *bioRxiv* 208819 (2017) doi:10.1101/208819.

8. Schlitzer, A. *et al.* Identification of cDC1- and cDC2-committed DC progenitors reveals early lineage priming at the common DC progenitor stage in the bone marrow. *Nature Immunology* **16**, 718–728 (2015).

9. Velten, L. *et al.* Human haematopoietic stem cell lineage commitment is a continuous process. *Nature Cell Biology* **19**, 271–281 (2017).

10. See, P. *et al.* Mapping the human DC lineage through the integration of high-dimensional techniques. *Science* **356**, eaag3009 (2017).

11. Aibar, S. *et al.* SCENIC: Single-cell regulatory network inference and clustering. *Nature Methods* **14**, 1083–1086 (2017).

12. Regev, A. *et al.* Science Forum: The Human Cell Atlas. *eLife* **6**, e27041 (2017).

13. Han, X. *et al.* Mapping the Mouse Cell Atlas by Microwell-Seq. *Cell* **172**, 1091–1107.e17 (2018).

14. Schaum, N. *et al.* Single-cell transcriptomics of 20 mouse organs creates a Tabula Muris. *Nature* **562**, 367–372 (2018).

15. Angerer, P. *et al.* Single cells make big data: New challenges and opportunities in transcriptomics. *Current Opinion in Systems Biology* **4**, 85–91 (2017).

16. Henry, V. J., Bandrowski, A. E., Pepin, A.-S., Gonzalez, B. J. & Desfeux, A. OMICtools: An informative directory for multi-omic data analysis. *Database: The Journal of Biological Databases and Curation* **2014**, (2014).

17. Davis, S. *et al.* https://github.com/seandavi/awesome-single-cell. (2018) doi:10.5281/zenodo.1294021.

18. Zappia, L., Phipson, B. & Oshlack, A. Exploring the single-cell RNA-seq analysis landscape with the scRNA-tools database. *bioRxiv* 206573 (2017) doi:10.1101/206573.

19. Bendall, S. C. *et al.* Single-Cell Trajectory Detection Uncovers Progression and Regulatory Coordination in Human B Cell Development. *Cell* **157**, 714–725 (2014).

20. Shin, J. *et al.* Single-Cell RNA-Seq with Waterfall Reveals Molecular Cascades underlying Adult Neurogenesis. *Cell Stem Cell* **17**, 360–372 (2015).

21. Campbell, K. & Yau, C. Bayesian Gaussian Process Latent Variable Models for pseudotime inference in single-cell RNA-seq data. *bioRxiv* 026872 (2015) doi:10.1101/026872.

22. Haghverdi, L., Büttner, M., Wolf, F. A., Buettner, F. & Theis, F. J. Diffusion pseudotime robustly reconstructs lineage branching. *Nature Methods* **13**, 845–848 (2016).

23. Setty, M. *et al.* Wishbone identifies bifurcating developmental trajectories from single-cell data. *Nature Biotechnology* **34**, 637–645 (2016).

24. Trapnell, C. *et al.* The dynamics and regulators of cell fate decisions are revealed by pseudotemporal ordering of single cells. *Nature Biotechnology* **32**, nbt.2859 (2014).

25. Matsumoto, H. & Kiryu, H. SCOUP: A probabilistic model based on the OrnsteinUhlenbeck process to analyze single-cell expression data during differentiation. *BMC Bioinformatics* **17**, 232 (2016).

26. Qiu, X. *et al.* Reversed graph embedding resolves complex single-cell trajectories. *Nature Methods* **14**, 979–982 (2017).

27. Street, K. *et al.* Slingshot: Cell lineage and pseudotime inference for single-cell transcriptomics. *BMC Genomics* **19**, 477 (2018).

28. Ji, Z. & Ji, H. TSCAN: Pseudo-time reconstruction and evaluation in single-cell RNA-seq analysis. *Nucleic Acids Research* **44**, e117–e117 (2016).

29. Welch, J. D., Hartemink, A. J. & Prins, J. F. SLICER: Inferring branched, nonlinear cellular trajectories from single cell RNA-seq data. *Genome Biology* **17**, 106 (2016).

30. duVerle, D. A., Yotsukura, S., Nomura, S., Aburatani, H. & Tsuda, K. CellTree: An R/bioconductor package to infer the hierarchical structure of cell populations from single-cell RNA-seq data. *BMC Bioinformatics* **17**, 363 (2016).

31. Cannoodt, R. *et al.* SCORPIUS improves trajectory inference and identifies novel modules in dendritic cell development. *bioRxiv* 079509 (2016) doi:10.1101/079509.

32. Lönnberg, T. *et al.* Single-cell RNA-seq and computational analysis using temporal mixture modeling resolves TH1/TFH fate bifurcation in malaria. *Science Immunology* **2**, eaal2192 (2017).

33. Campbell, K. R. & Yau, C. Probabilistic modeling of bifurcations in single-cell gene expression data using a Bayesian mixture of factor analyzers. *Wellcome Open Research* **2**, 19 (2017).

34. Tian, L. *et al.* scRNA-seq mixology: Towards better benchmarking of single cell RNA-seq protocols and analysis methods. *bioRxiv* 433102 (2018) doi:10.1101/433102.

35. Schaffter, T., Marbach, D. & Floreano, D. GeneNetWeaver: In silico benchmark generation and performance profiling of network inference methods. *Bioinformatics (Oxford, England)* **27**, 2263–2270 (2011).

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

37. Svensson, V., Vento-Tormo, R. & Teichmann, S. A. Exponential scaling of single-cell RNA-seq in the past decade. *Nature Protocols* **13**, 599–604 (2018).

38. Cao, J. *et al.* Joint profiling of chromatin accessibility and gene expression in thousands of single cells. *Science* eaau0730 (2018) doi:10.1126/science.aau0730.

39. Pya, N. & Wood, S. N. Shape constrained additive models. *Statistics and Computing* **25**, 543–559 (2015).

40. Taschuk, M. & Wilson, G. Ten simple rules for making research software more robust. *PLOS Computational Biology* **13**, e1005412 (2017).

41. Mangul, S. *et al.* A comprehensive analysis of the usability and archival stability of omics computational tools and resources. *bioRxiv* 452532 (2018) doi:10.1101/452532.

42. Wilson, G. *et al.* Best Practices for Scientific Computing. *PLOS Biology* **12**, e1001745 (2014).

43. Artaza, H. *et al.* Top 10 metrics for life science software good practices. *F1000Research* **5**, 2000 (2016).

44. Saelens, W., Cannoodt, R. & Saeys, Y. A comprehensive evaluation of module detection methods for gene expression data. *Nature Communications* **9**, 1090 (2018).

45. Manno, G. L. *et al.* RNA velocity of single cells. *Nature* **560**, 494–498 (2018).

46. Norel, R., Rice, J. J. & Stolovitzky, G. The self-assessment trap: Can we all be better than average? *Molecular Systems Biology* **7**, 537 (2011).

47. Cannoodt, R., Saelens, W., Todorov, H. & Saeys, Y. Single-cell -omics datasets for containing a trajectory. *Zenodo* doi:10.5281/zenodo.1443566.

48. Gitter, A. https://github.com/agitter/single-cell-pseudotime. (2018) doi:10.5281/zenodo.1297423.

49. Kouno, T. *et al.* Temporal dynamics and transcriptional control using single-cell gene expression analysis. *Genome Biology* **14**, R118 (2013).

50. Zeng, C. *et al.* Pseudotemporal Ordering of Single Cells Reveals Metabolic Control of Postnatal β Cell Proliferation. *Cell Metabolism* **25**, 1160–1175.e11 (2017).

51. Papadopoulos, N., Parra, R. G. & Soeding, J. PROSSTT: Probabilistic simulation of single-cell RNA-seq data for complex differentiation processes. *bioRxiv* 256941 (2018) doi:10.1101/256941.

52. Marbach, D. *et al.* Wisdom of crowds for robust gene network inference. *Nature methods* **9**, 796–804 (2012).

53. Xu, H. *et al.* Regulation of bifurcating B cell trajectories by mutual antagonism between transcription factors IRF4 and IRF8. *Nature Immunology* **16**, 1274–1281 (2015).

54. Graf, T. & Enver, T. Forcing cells to change lineages. *Nature* **462**, 587 (2009).

55. Wang, J., Zhang, K., Xu, L. & Wang, E. Quantifying the Waddington landscape and biological paths for development and differentiation. *Proceedings of the National Academy of Sciences* **108**, 8257–8262 (2011).

56. Ferrell, J. E. Bistability, Bifurcations, and Waddington's Epigenetic Landscape. *Current Biology* **22**, R458–R466 (2012).

57. Yosef, N. *et al.* Dynamic regulatory network controlling Th17 cell differentiation. *Nature* **496**, 461–468 (2013).

58. Marbach, D. *et al.* Tissue-specific regulatory circuits reveal variable modular perturbations across complex diseases. *Nature Methods* **13**, 366 (2016).

59. Giorgino, T. Computing and Visualizing Dynamic Time Warping Alignments in R: The dtw Package. *Journal of Statistical Software* **31**, (2009).

60. Tormene, P., Giorgino, T., Quaglini, S. & Stefanelli, M. Matching incomplete time series with dynamic time warping: An algorithm and an application to post-stroke rehabilitation. *Artificial Intelligence in Medicine* **45**, 11–34 (2009).

61. Lun, A. T. L., McCarthy, D. J. & Marioni, J. C. A step-by-step workflow for low-level analysis of single-cell RNA-seq data with Bioconductor. *F1000Research* **5**, 2122 (2016).

62. Jurman, G., Visintainer, R., Filosi, M., Riccadonna, S. & Furlanello, C. The HIM glocal metric and kernel for network comparison and classification. in *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)* 1–10 (2015). doi:10.1109/DSAA.2015.7344816.

63. Wright, M. N. & Ziegler, A. Ranger: A Fast Implementation of Random Forests for High Dimensional Data in C++ and R | Wright | Journal of Statistical Software. *Journal of Statistical Software* **77**, (2017).

64. Beaulieu-Jones, B. K. & Greene, C. S. Reproducibility of computational workflows is automated using continuous analysis. *Nature Biotechnology* **35**, nbt.3780 (2017).