nature biotechnology

ARTICLES

https://doi.org/10.1038/s41587-019-0071-9

A comparison of single-cell trajectory inference methods

Wouter Saelens[1,2,6], Robrecht Cannoodt[1,3,4,6], Helena Todorov[1,2,5] and Yvan Saeys[1,2*]

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

Single-cell omics data, including transcriptomics, proteomics and epigenomics data, provide new opportunities for studying cellular dynamic processes, such as the cell cycle, cell differentiation and cell activation[1,2]. Such dynamic processes can be modeled computationally using trajectory inference (TI) methods, also called pseudotime analysis, which order cells along a trajectory based on similarities in their expression patterns[3–5]. The resulting trajectories are most often linear, bifurcating or tree-shaped, but more recent methods also identify more complex trajectory topologies, such as cyclic[6] or disconnected graphs[7]. TI methods offer an unbiased and transcriptome-wide understanding of a dynamic process[1], thereby allowing the objective identification of new (primed) subsets of cells[8], delineation of a differentiation tree[9,10] and inference of regulatory interactions responsible for one or more bifurcations[11]. Current applications of TI focus on specific subsets of cells, but ongoing efforts to construct transcriptomic catalogs of whole organisms[12–14] underline the urgency for accurate, scalable[11,15] and user-friendly TI methods.

A plethora of TI methods has been developed over the past few years and even more are being created every month (Supplementary Table 1). Indeed, in several repositories listing single-cell tools, such as omictools.org[16], the 'awesome-single-cell' list[17] and scRNA-tools.org[18], TI methods are one of the largest categories. While each method has its own unique set of characteristics in terms of underlying algorithm, required prior information and produced outputs, two of the most distinctive differences between TI methods are whether they fix the topology of the trajectory and what type(s) of graph topologies they can detect. Early TI methods typically fixed the topology algorithmically (for example, linear[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. More recent methods also infer the topology[7,26,27], which increases the difficulty of the problem at hand, but allows the unbiased identification of both the ordering inside a branch and the topology connecting these branches.

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

In this study, we evaluated the accuracy, scalability, stability and usability of 45 TI methods (Fig. 1a). We found substantial complementarity between current methods, with different sets of methods performing most optimally depending on the characteristics of the data. For method users, we created an interactive set of guidelines (available at guidelines.dynverse.org), which gives context-specific recommendations for method usage. Our evaluation also highlights some challenges for current methods, and our evaluation strategy can be useful to spearhead the development of new tools that accurately infer trajectories on ever more complex use cases.

Results

Trajectory inference methods. To make the outputs from different methods directly comparable to each other, we developed a common probabilistic model for representing trajectories from all possible sources (Fig. 1b). In this model, the overall topology is represented by a network of 'milestones', and the cells are placed within the space formed by each set of connected milestones. Although almost every method returned a unique set of outputs, we were able to classify these outputs into seven distinct groups (Supplementary Fig. 1) and we wrote a common output converter for each of these groups (Fig. 2a). When strictly required, we also provided prior information to the method. These different priors can range from weak priors that are relatively easy to acquire, such as a start cell, to strong priors, such as a known grouping of cells, that are much

[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]Department of Biomolecular Medicine, Ghent University, Ghent, Belgium. [5]Centre International de Recherche en Infectiologie, Inserm, U1111, Université Claude Bernard Lyon 1, CNRS, UMR5308, École Normale Supérieure de Lyon, Université de Lyon, Lyon, France. [6]These authors contributed equally: Wouter Saelens, Robrecht Cannoodt. *e-mail: yvan.saeys@ugent.be

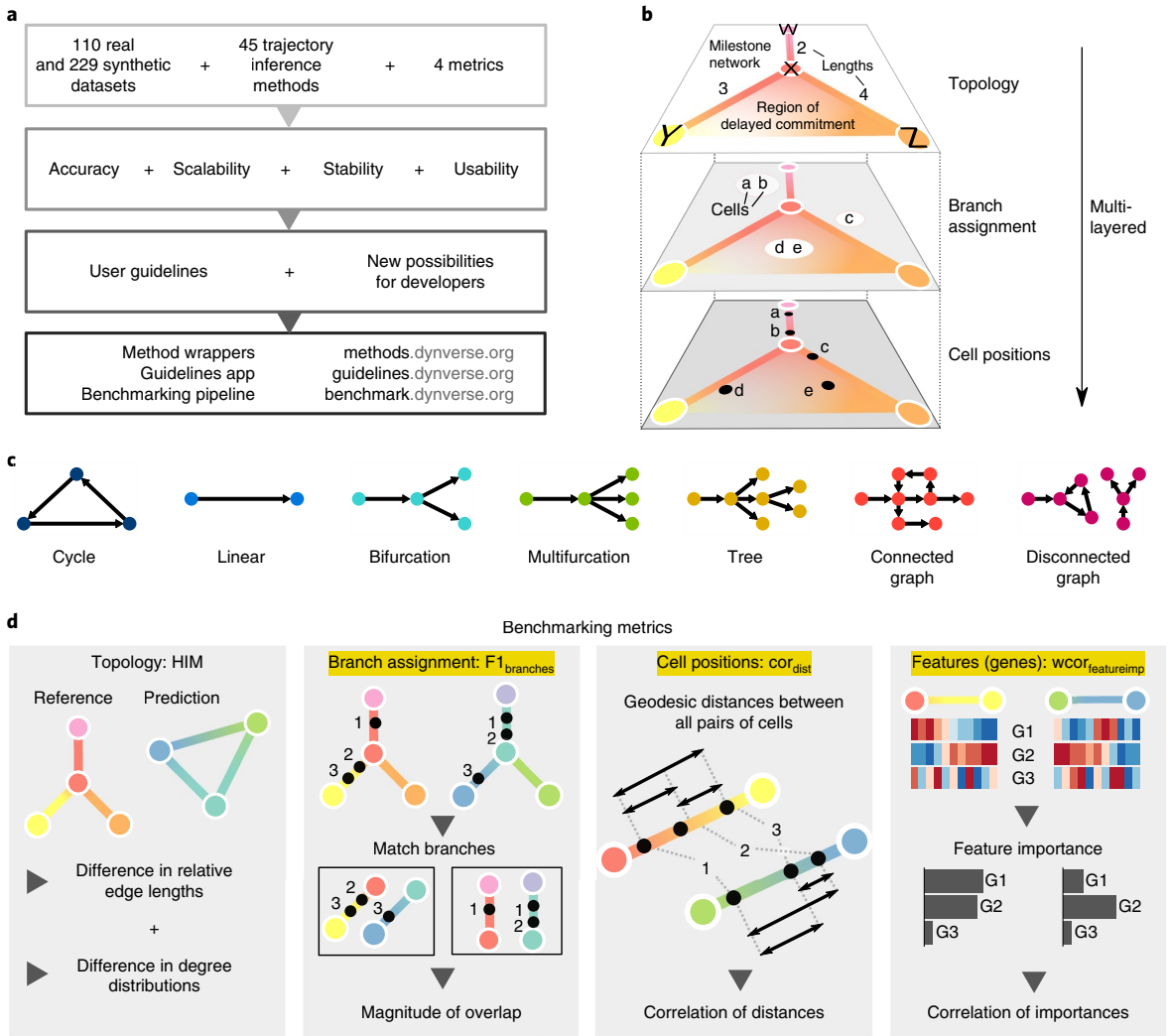

Fig. 1 | Overview of several key aspects of the evaluation. a, A schematic overview of our evaluation pipeline. **b**, To make the trajectories comparable to each other, a common trajectory model was used to represent reference trajectories from the real and synthetic datasets, as well as any predictions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

harder to know a priori, and which can potentially introduce a large bias into the analysis (Fig. 2a).

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

We evaluated each method on four core aspects: (1) accuracy of a prediction, given a gold or silver standard on 110 real and 229 synthetic datasets; (2) scalability with respect to the number of cells and features (for example, genes); (3) stability of the predictions after subsampling the datasets; and (4) the usability of the tool in terms of software, documentation and the manuscript. Overall, we found a large diversity across the four evaluation criteria, with only a few methods, such as PAGA, Slingshot and SCORPIUS, performing well across the board (Fig. 2b). We will discuss each evaluation criterion in more detail (Fig. 3 and Supplementary Fig. 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 (Fig. 1d): the topology (Hamming–Ipsen–Mikhailov, HIM), the quality of the assignment of cells to branches ($F1_{branches}$), the cell positions (cor_{dist}) and the accuracy of the differentially expressed features along the trajectory ($wcor_{features}$). The data compendium consisted of both synthetic datasets, which offer the most exact reference trajectory, and real datasets, which provide the highest biological relevance. These real datasets come from a variety of single-cell technologies, organisms and dynamic processes, and contain several types of trajectory topologies (Supplementary Table 2).

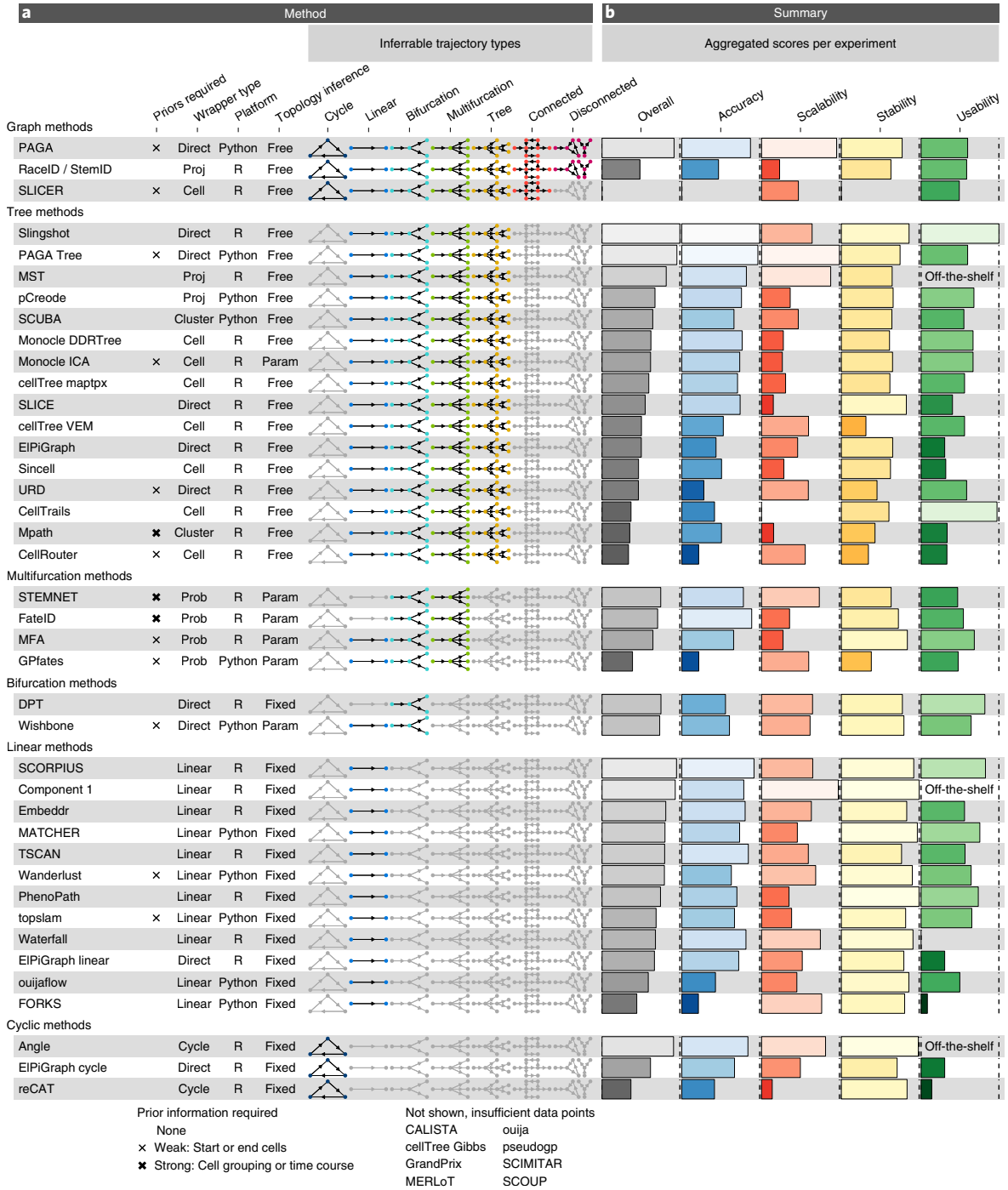

Fig. 2 | A characterization of the 45 methods evaluated in this study and their overall evaluation results. a, We characterized the methods according to the wrapper type, their required priors, whether the inferred topology is constrained by the algorithm (fixed) or a parameter (param), and the types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

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 Fig. 3a). Even methods that can detect most of the trajectory types, such as PAGA, RaceID/StemID and SLICER were not the best methods across all

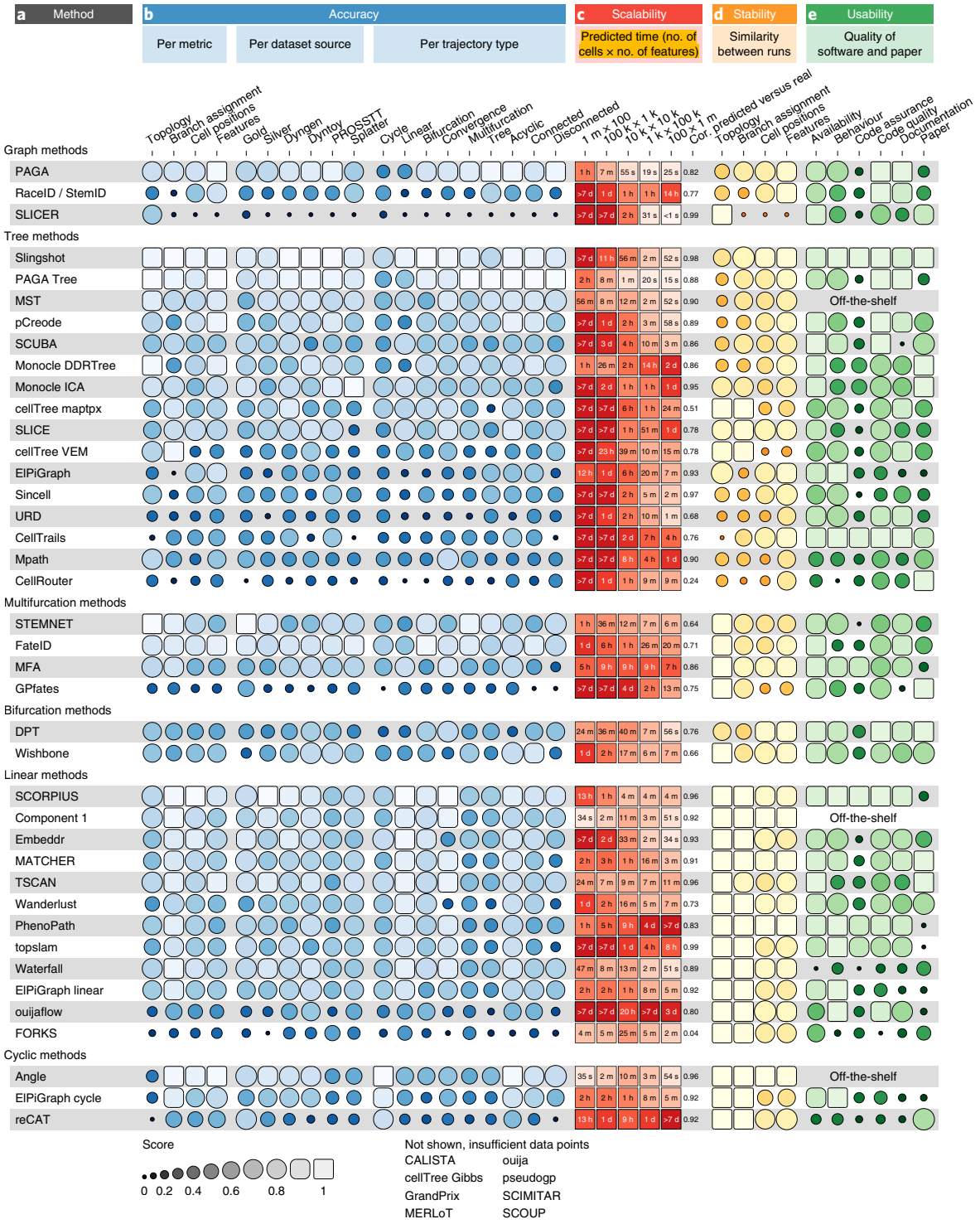

Fig. 3 | Detailed results of the four main evaluation criteria: accuracy, scalability, stability and usability. a, The names of the methods, ordered as in Fig. 2. **b**, Accuracy of trajectory inference methods across metrics, dataset sources and dataset trajectory types. The performance of a method 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 (no. of cells × no. of features). Predictions were made by training a regression model after running each method on bootstrapped datasets with varying numbers of cells and features. k, thousands; m, millions; cor, correlation. **d**, Stability results by calculating the average pairwise similarity between models inferred across multiple runs of the same method. **e**, Usability scores of the tool and corresponding manuscript, grouped per category. Off-the-shelf methods were directly implemented in R and thus do not have a usability score.

trajectory types (Fig. 3b). The overall score between the different dataset sources was moderately to highly correlated (Spearman rank correlation between 0.5–0.9) with the scores on real datasets containing a gold standard (Supplementary Fig. 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, whereas other methods, such

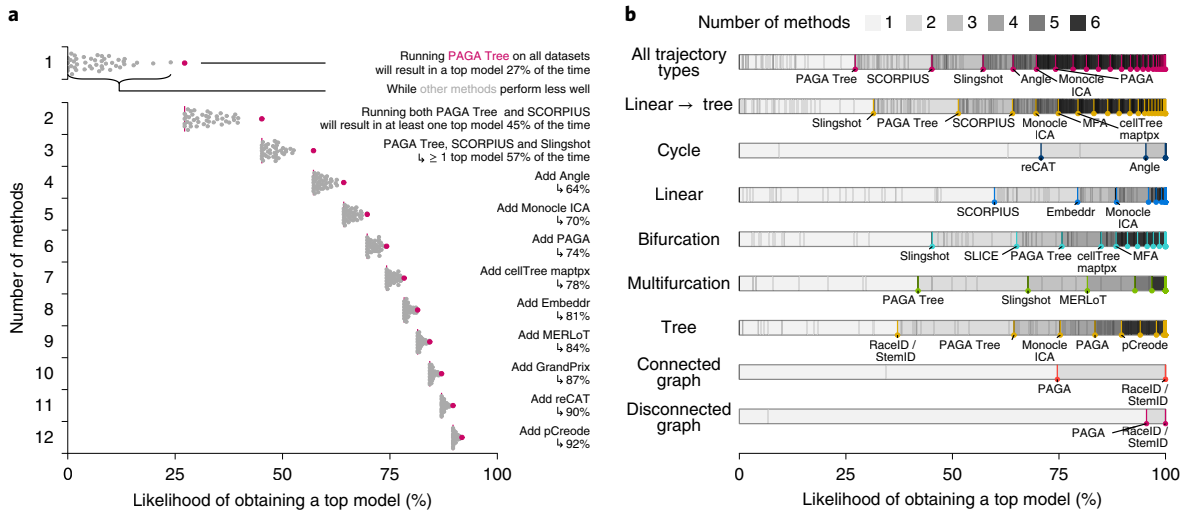

Fig. 4 | Complementarity between different trajectory inference methods. a, We assessed the likelihood for different combinations of methods to lead to a 'top model' (defined as a model with an overall score of at least 95% of the best model) when applied to all datasets. **b**, The likelihood for different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

as Slingshot, were better at ordering the cells and placing them into the correct branches (Fig. 3b).

The performance of a method was strongly dependent on the type of trajectory present in the data (Fig. 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 Fig. 3c). This was reflected in the types of topologies detected by every method, as those predicted by Slingshot tended to contain less branches, whereas those detected by PAGA, pCreode and Monocle DDRTree gravitated towards more complex topologies (Supplementary Fig. 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 resulted in getting a top model about 27% of the time. This increased up to 74% with the addition of six other methods (Fig. 4a). The result was a relatively diverse set of methods, containing both strictly linear or cyclic methods, and methods with a broad trajectory type range such as PAGA. We found similar indications of complementarity between the top methods on data containing only linear, bifurcation or multifurcating trajectories (Fig. 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 that utilize this complementarity.

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

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

Methods with a low running time typically had two defining aspects: they had a linear time complexity with respect to the features and/or cells, and adding new cells or features led to a relatively low increase in time (Supplementary Fig. 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 Fig. 4b).

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

Altogether, the scalability analysis indicated that the dimensions of the data are an important factor in the choice of method, and

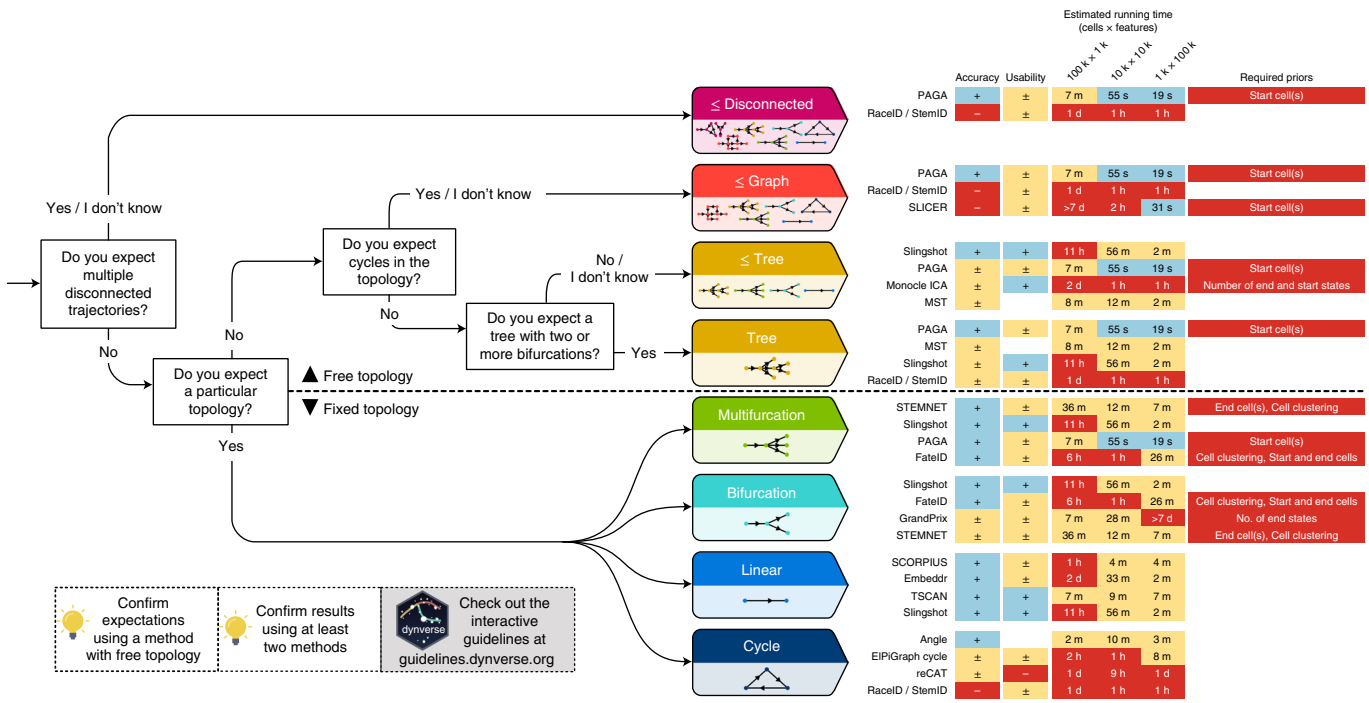

Fig. 5 | Practical guidelines for method users. As the performance of a method mostly depends on the topology of the trajectory, the choice of TI method will be primarily influenced by the user's existing knowledge about the expected topology in the data. We therefore devised a set of practical guidelines, 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. k, thousands; m, millions.

that method development should pay more attention to maintaining reasonable running times and memory usage.

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

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

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

We found that most methods fulfilled the basic criteria, such as the availability of a tutorial and elemental code quality criteria (Fig. 3d and Supplementary Fig. 6). While recent methods had a slightly better quality score than older methods, several quality aspects were consistently lacking for the majority of the methods (Supplementary Fig. 6 right) and we believe that these should receive extra attention from developers. Although these outstanding issues covered all five categories, code assurance and documentation in particular were problematic areas, notwithstanding several studies pinpointing these as good practices[42,43]. Only two methods had a nearly perfect usability score (Slingshot and Celltrails), and these could be used as an inspiration for future methods. We observed no clear relation between usability and method accuracy or usability (Fig. 2b).

Discussion

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

Based on the results of our benchmark, we propose a set of practical guidelines for method users (Fig. 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 most use cases, the user will know very little about the expected trajectory, except perhaps whether the data is expected to contain multiple disconnected trajectories, cycles or a complex tree structure. In each of these use cases, our evaluation

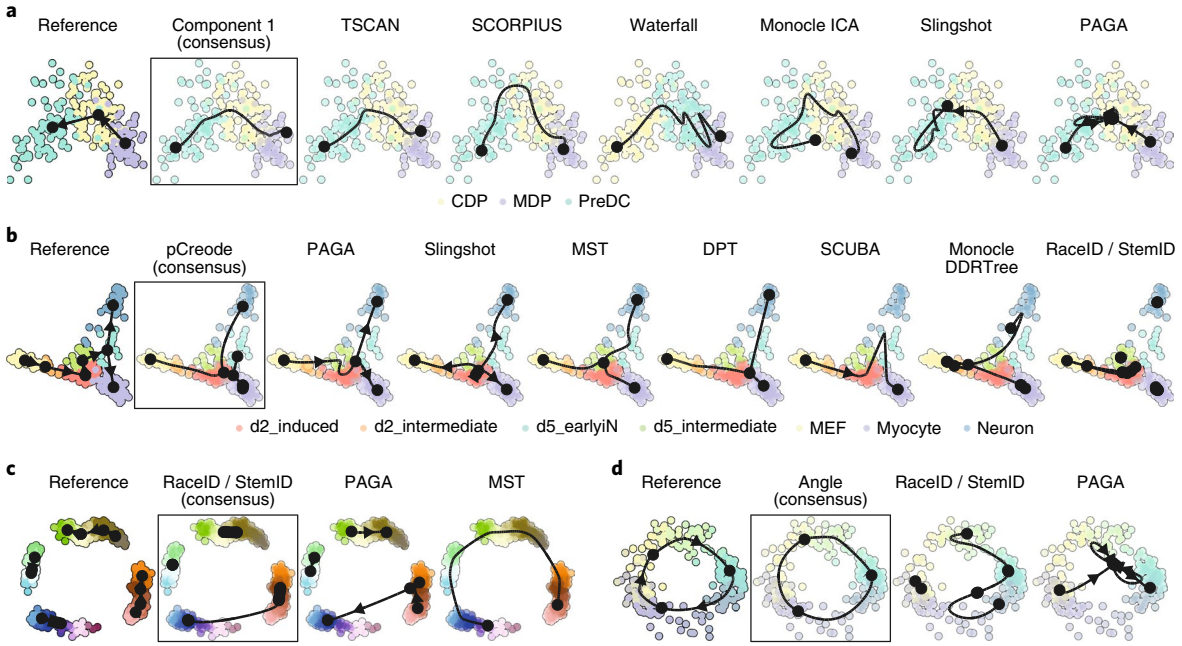

Fig. 6 | Demonstration of how a common framework for TI methods facilitates broad applicability using some example datasets. Trajectories inferred by each method were projected to a common dimensionality reduction using multidimensional scaling. For each dataset, we also calculated a 'consensus' prediction,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

suggests a different set of optimal methods, as shown in Fig. 5. Several other factors will also impact the choice of methods, such as the dimensions of the dataset and the prior information that is available. These factors and several others can all be dynamically explored in our interactive app (guidelines.dynverse.org). This app can also be used to query the results of this evaluation, such as filtering the datasets or changing the importance of the evaluation metrics for the final ranking.

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

Critical to the broad applicability of TI methods is the standardization of the input and output interfaces of TI methods, so that users can effortlessly execute TI methods on their dataset of interest, 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 (Fig. 6). Using our framework, this figure can be recreated using only a couple of lines of R code (https://methods.dynverse.org). In the future, this framework could be extended to allow additional input data, such as spatial and RNA velocity information[45], and easier downstream analyses. In addition, further discussion within the field is required to arrive at a consensus concerning a common interface for trajectory models, which can include additional features such as uncertainty and gene importance.

Our study indicates that the field of trajectory inference is maturing, primarily for linear and bifurcating trajectories (Fig. 6a,b). However, we also highlight several ongoing challenges, which should be addressed before TI can be a reliable tool for analyzing single-cell omics datasets with complex trajectories. Foremost, new methods should focus on improving the unbiased inference of tree, cyclic graph and disconnected topologies, as we found that methods repeatedly overestimate or underestimate the complexity of the underlying topology, even if the trajectory could easily be identified using a dimensionality reduction method (Fig. 6c,d). Furthermore, higher standards for code assurance and documentation could help in adopting these tools across the single-cell omics field. Finally, new tools should be designed to scale well with the increasing number of cells and features. We found that only a handful of current methods can handle datasets with more than 10,000 cells within a reasonable time frame. To support the development of these new tools, we provide a series of vignettes on how to wrap and evaluate a method on the different measures proposed in this study at https://benchmark.dynverse.org.

We found that the performance of a method can be very variable between datasets, and therefore included a large set of both real and synthetic data within our evaluation, leading to a robust overall ranking of the different methods. However, 'good-yet-not-the-best' methods[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.

Online content

Any methods, additional references, Nature Research reporting summaries, source data, statements of data availability, and associated accession codes are available at https://doi.org/10.1038/s41587-019-0071-9.

Received: 5 April 2018; Accepted: 13 February 2019;
Published online: 1 April 2019

References

1. Tanay, A. & Regev, A. Scaling single-cell genomics from phenomenology to mechanism. *Nature* **541**, 21350 (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 Res.* **25**, 1491–1498 (2015).
4. Cannoodt, R., Saelens, W. & Saeys, Y. Computational methods for trajectory inference from single-cell transcriptomics. *Eur. J. Immunol.* **46**, 2496–2506 (2016).
5. Moon, K. R. et al. Manifold learning-based methods for analyzing single-cell RNA-sequencing data. *Curr. Opin. Syst. Biol.* **7**, 36–46 (2018).
6. Liu, Z. et al. Reconstructing cell cycle pseudo time-series via single-cell transcriptome data. *Nat. Commun.* **8**, 22 (2017).
7. Wolf, F. A. et al. PAGA: graph abstraction reconciles clustering with trajectory inference through a topology preserving map of single cells. *Genome Biol.* **20**, 59 (2019).
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. *Nat. Immunol.* **16**, 718–728 (2015).
9. Velten, L. et al. Human haematopoietic stem cell lineage commitment is a continuous process. *Nat. Cell Biol.* **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. *Nat. 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. *Curr. Opin. Syst. Biol.* **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 (Oxford)* **2014**, bau069 (2014).
17. Davis, S. et al. List of software packages for single-cell data analysis. https://github.com/seandavi/awesome-single-cell (2018); https://doi.org/10.5281/zenodo.1294021
18. Zappia, L., Phipson, B. & Oshlack, A. Exploring the single-cell RNA-seq analysis landscape with the scRNA-tools database. *PLoS Comput. Biol.* **14,** e1006245 (2018)
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. Preprint at *bioRxiv* https://doi.org/10.1101/026872 (2015).
22. Haghverdi, L., Büttner, M., Wolf, F. A., Buettner, F. & Theis, F. J. Diffusion pseudotime robustly reconstructs lineage branching. *Nat. Methods* **13**, 845–848 (2016).
23. Setty, M. et al. Wishbone identifies bifurcating developmental trajectories from single-cell data. *Nat. Biotechnol.* **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. *Nat. Biotechnol.* **32**, 2859 (2014).
25. Matsumoto, H. & Kiryu, H. SCOUP: a probabilistic model based on the Ornstein–Uhlenbeck 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. *Nat. 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 Res.* **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. Biol.* **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. Preprint at *bioRxiv* https://doi.org/10.1101/079509 (2016).
32. Lönnberg, T. et al. Single-cell RNA-seq and computational analysis using temporal mixture modeling resolves TH1/TFH fate bifurcation in malaria. *Sci. Immunol.* **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 Res.* **2**, 19 (2017).
34. Tian, L. *et al.* scRNA-seq mixology: Towards better benchmarking of single cell RNA-seq protocols and analysis methods. Preprint at *bioRxiv* https://doi.org/10.1101/433102 (2018).
35. Schaffter, T., Marbach, D. & Floreano, D. GeneNetWeaver: in silico benchmark generation and performance profiling of network inference methods. *Bioinformatics* **27**, 2263–2270 (2011).
36. Zappia, L., Phipson, B. & Oshlack, A. Splatter: simulation of single-cell RNA sequencing data. *Genome. Biol.* **18**, 174 (2017).
37. Svensson, V., Vento-Tormo, R. & Teichmann, S. A. Exponential scaling of single-cell RNA-seq in the past decade. *Nat. Protoc.* **13**, 599–604 (2018).
38. Cao, J. *et al.* Joint profiling of chromatin accessibility and gene expression in thousands of single cells. *Science* **361**, 1380–1385 (2018).
39. Pya, N. & Wood, S. N. Shape constrained additive models. *Stat. Comput.* **25**, 543–559 (2015).
40. Taschuk, M. & Wilson, G. Ten simple rules for making research software more robust. *PLoS Comput. Biol.* **13**, e1005412 (2017).
41. Mangul, S. et al. A comprehensive analysis of the usability and archival stability of omics computational tools and resources. Preprint at *bioRxiv* https://doi.org/10.1101/452532 (2018).
42. Wilson, G. et al. Best practices for scientific computing. *PLoS Biol.* **12**, e1001745 (2014).
43. Artaza, H. et al. Top 10 metrics for life science software good practices. *F1000Res.* **5**, 2000 (2016).
44. Saelens, W., Cannoodt, R. & Saeys, Y. A comprehensive evaluation of module detection methods for gene expression data. *Nat. Commun.* **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? *Mol. Syst. Biol.* **7**, 537 (2011).

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 (R.C., 11Y6218N and W.S., 11Z4518N) and BOF (Ghent University, H.T.). Y.S. is an ISAC Marylou Ingram scholar.

Author contributions

R.C., W.S., H.T. and Y.S. designed the study. R.C. and W.S. performed the experiments and analyzed 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 interests

The authors declare no competing interests.

Additional information

Supplementary information is available for this paper at https://doi.org/10.1038/s41587-019-0071-9.

Reprints and permissions information is available at www.nature.com/reprints.

Correspondence and requests for materials should be addressed to Y.S.

Publisher's note: Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Methods

Trajectory inference methods. We gathered a list of 71 trajectory inference tools (Supplementary Table 1) by searching the literature for 'trajectory inference' and 'pseudotemporal ordering', and based on two existing lists found online: https://github.com/seandavi/awesome-single-cell[17] and https://github.com/agitter/single-cell-pseudotime[47]. We welcome any contributions by creating an issue at https://methods.dynverse.org.

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

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

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

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

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

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

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

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

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

Special conversions were necessary for certain methods:

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

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

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

- **Component 1:** This method returns the first component of a principal component analysis (PCA) dimensionality reduction as a linear trajectory. This method is especially relevant as it has been used in a few studies already[48,49].
- **Angle:** Similar to the previous method, this method computes the angle with respect to the origin in a two-dimensional PCA and uses this angle as a pseudotime for generating a cyclical trajectory.
- **MST:** This method performs PCA dimensionality reduction, followed by clustering using the R mclust package, after which the clusters are connected using a minimum spanning tree. The trees are orthogonally projected to the nearest segment of the tree. This baseline is highly relevant as many methods follow the same methodology: dimensionality reduction, clustering, topology inference and project cells to topology.

Trajectory types. We classified all possible trajectory topologies into distinct trajectory types, based on topological criteria (Fig. 1c). These trajectory types start from the most general trajectory type, a disconnected graph, and move down (within a directed acyclic graph structure), progressively becoming more simple until the two basic types are reached: linear and cyclical. A disconnected graph is a graph in which only one edge can exist between two nodes. A (connected) graph is a disconnected graph in which all nodes are connected. An acyclic graph is a graph containing no cycles. A tree is an acyclic graph containing no convergences (no nodes with in-degree higher than 1). A convergence is an acyclic graph in which only one node has a degree larger than 1 and this same node has an in-degree of 1. A multifurcation is a tree in which only one node has a degree larger than 1. A bifurcation is a multifurcation in which only one node has a degree equal to 3. A linear topology is a graph in which no node has a degree larger than 3. Finally, a cycle is a connected graph in which every node has a degree equal to 2. In most cases, a method that was able to detect a complex trajectory type was also able to detect less complex trajectory types, with some exceptions shown in Fig. 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 (https://benchmark.dynverse.org/tree/master/scripts/01-datasets). Whenever possible, we preferred to start from the raw counts data. These raw counts were all normalized and filtered using a common pipeline, as discussed later. Some original datasets contained more than one trajectory, in which case we split the dataset into its separate connected trajectory, but also generated several combinations of connected trajectories to include some datasets with disconnected trajectories in the evaluation. In the end, we included 110 datasets for this evaluation.

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

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

- **dyngen:** simulations of gene regulatory networks, available at https://github.com/dynverse/dyngen
- **dyntoy:** random gradients of expression in the reduced space, available at https://github.com/dynverse/dyntoy
- **PROSSTT:** expression is sampled from a linear model that depends on pseudotime[50]
- **Splatter:** simulations of non-linear paths between different expression states[36]

These simulators are discussed in Supplementary Note 2.

Dataset filtering and normalization. We used a standard single-cell RNA-seq preprocessing pipeline that applies parts of the scran and scater Bioconductor packages[51]. The advantages of this pipeline are that it works both with and without spike-ins, and it includes a harsh cell filtering that looks at abnormalities in library sizes, mitochondrial gene expression and the number of genes expressed using median absolute deviations (which we set to 3). We required that a gene was expressed in at least 5% of the cells and that it should have an average expression higher than 0.02. Furthermore, we used the pipeline to select the most highly variable genes, using a false discovery rate of 5% and a biological component higher than 0.5. As a final filter, we removed both all-zero genes and cells until convergence.

Benchmark metrics. The importance of using multiple metrics to compare complex models has been stated repeatedly[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 four metrics for the evaluation, each assessing a different aspect of the trajectory: (1) the HIM measures the topological similarity; (2) the $F1_{branches}$ compares the branch assignment; (3) the cor_{dist} assesses the similarity in pairwise cell–cell distances and thus the cellular positions; and (4) the $wcor_{features}$ looks at whether similar important features (genes) are found in both the reference dataset and the prediction.

The Hamming–Ipsen–Mikhailov metric. The HIM metric[52] uses the two weighted adjacency matrices of the milestone networks as input (weighted by edge length). It is a linear combination of the normalized Hamming distance, which gives an indication of the differences in edge lengths, and the normalized Ipsen–Mikhailov distance, which assesses the similarity in degree distributions. The latter has a parameter γ, which was fixed at 0.1 to make the scores comparable between datasets. We illustrate the metric and discuss alternatives in Supplementary Note 1.

The F1 between branch assignments. To compare branch assignment, we used an *F1* score, also used used for comparing biclustering methods[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 defined the 'Recovery' (respectively 'Relevance') as the average maximal similarity of all branches in the reference dataset (respectively prediction). The $F1_{branches}$ was then defined as the harmonic mean between Recovery and Relevance. We illustrate this metric further in Supplementary Note 1.

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

The correlation between important features. The $wcor_{features}$ assesses whether the same differentially expressed features are found using the predicted trajectory as in the known trajectory. To calculate this metric, we used Random Forest regression (implemented in the R ranger package[53]), to predict expression values of each gene, based on the geodesic distances of a cell to each milestone. We then extracted feature importance values for each feature and calculated the similarity of the feature importances using a weighted Pearson correlation, weighted by the feature importance in the reference dataset to give more weight to large differences. As hyperparameters we set the number of trees to 10,000 and the number of features on which to split to 1% of all available features. We illustrate this metric and assess the effect of its hyperparameters in Supplementary Note 1.

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

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

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

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

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

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

Scalability. To assess the scalability of each method, we started from five real datasets, selected using the centers from a *k*-medoids as discussed in Supplementary Note 2. We up- and downscaled these datasets between 10 and 100,000 cells and 10 and 100,000 features, while never going higher than 1,000,000 values in total. To generate new cells or features, we first generated a 10-nearest-neighbor graph of both the cells and features from the expression space. For every new cell or feature, we used a linear combination of one to three existing cells or features, where each cell or feature was given a weight sampled from a uniform distribution between 0 and 1.

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

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

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

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

This process was repeated for the classification of 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 $\text{F1}_{\text{branches}}$, we only used the intersection between the cells of two datasets, while the intersection of the features was used for the $\text{wcor}_{\text{features}}$.

Usability. We created a transparent scoring scheme to quantify the usability of each method based on several existing tool quality and programming guidelines in the literature and online (Supplementary Table 3). The main goal of this quality control is to stimulate the improvement of current methods, and the development of user- and developer-friendly new methods. The quality control assessed six categories, each looking at several aspects, which are further divided into individual items. The availability category checks whether the method is easily available, whether the code and dependencies can be easily installed, and how the method can be used. The code quality assesses the quality of the code both from a user perspective (function naming, dummy proofing and availability of plotting functions) and a developer perspective (consistent style and code duplication). The code assurance category is frequently overlooked, and checks for code testing, continuous integration[54] and an active support system. The documentation category checks the quality of the documentation, both externally (tutorials and function documentation) and internally (inline documentation). The behavior category assesses the ease by which the method can be run, by looking for unexpected output files and messages, prior information and how easy the trajectory model can be extracted from the output. Finally, we also assessed certain aspects of the study in which the method was proposed, such as publication in a peer-reviewed journal, the number of datasets in which the usefulness of the method was shown and the scope of method evaluation in the paper.

Each quality aspect received a weight depending on how frequently it was found in several papers and online sources that discuss tool quality (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 that can detect all required trajectory types, and ordering the methods according to their average accuracy score on datasets containing these trajectory types (aggregated according to the scheme presented in the section Accuracy).

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

Reporting Summary. Further information on research design is available in the Nature Research Reporting Summary linked to this article.

Data availability

The processed real and synthetic datasets used in this study are deposited on Zenodo (https://doi.org/10.5281/zenodo.1443566)[55].
The main analysis repository is available at https://benchmark.dynverse.org and is divided into several experiments. Every experiment has its own set of scripts and results, each accompanied by an illustrated readme that can be browsed and explored on the Github website.

Code availability

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

References

47. Gitter, A. Single-cell RNA-seq pseudotime estimation algorithms. https://github.com/agitter/single-cell-pseudotime (2018); https://doi.org/10.5281/zenodo.1297423
48. Kouno, T. et al. Temporal dynamics and transcriptional control using single-cell gene expression analysis. *Genome. Biol.* **14**, R118 (2013).
49. Zeng, C. et al. Pseudotemporal ordering of single cells reveals metabolic control of postnatal β cell proliferation. *Cell. Metab.* **25**, 1160–1175.e11 (2017).
50. Papadopoulos, N., Parra, R. G. & Soeding, J. PROSSTT: probabilistic simulation of single-cell RNA-seq data for complex differentiation processes. *Bioinformatics,* btz078 (2019).

51. Lun, A. T., McCarthy, D. J. & Marioni, J. C. A step-by-step workflow for low-level analysis of single-cell RNA-seq data with Bioconductor. *F1000Res.* **5**, 2122 (2016).
52. Jurman, G., Visintainer, R., Filosi, M., Riccadonna, S. & Furlanello, C. in *Proc. 2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)* 1–10 (IEEE, 2015); https://doi.org/10.1109/DSAA.2015.7344816
53. Wright, M. N. & Ziegler, A. Ranger: a fast implementation of random forests for high dimensional data in C++ and R. *J. Stat. Softw.* **77**, 1-17 (2017).
54. Beaulieu-Jones, B. K. & Greene, C. S. Reproducibility of computational workflows is automated using continuous analysis. *Nat. Biotechnol.* **35**, 3780 (2017).
55. Cannoodt, R., Saelens, W., Todorov, H. & Saeys, Y. Single-cell -omics datasets containing a trajectory (Version 2.0.0). *Zenodo* https://doi.org/10.5281/zenodo.1443566 (2018).

nature research

Corresponding author(s): Yvan Saeys

Reporting Summary

Nature Research wishes to improve the reproducibility of the work that we publish. This form provides structure for consistency and transparency in reporting. For further information on Nature Research policies, see Authors & Referees and the Editorial Policy Checklist.

Statistical parameters

When statistical analyses are reported, confirm that the following items are present in the relevant location (e.g. figure legend, table legend, main text, or Methods section).

n/a	Confirmed	
☒	☐	The exact sample size (*n*) for each experimental group/condition, given as a discrete number and unit of measurement
☒	☐	An indication of whether measurements were taken from distinct samples or whether the same sample was measured repeatedly
☐	☒	The statistical test(s) used AND whether they are one- or two-sided *Only common tests should be described solely by name; describe more complex techniques in the Methods section.*
☒	☐	A description of all covariates tested
☒	☐	A description of any assumptions or corrections, such as tests of normality and adjustment for multiple comparisons
☒	☐	A full description of the statistics including central tendency (e.g. means) or other basic estimates (e.g. regression coefficient) AND variation (e.g. standard deviation) or associated estimates of uncertainty (e.g. confidence intervals)
☒	☐	For null hypothesis testing, the test statistic (e.g. *F*, *t*, *r*) with confidence intervals, effect sizes, degrees of freedom and *P* value noted *Give P values as exact values whenever suitable.*
☒	☐	For Bayesian analysis, information on the choice of priors and Markov chain Monte Carlo settings
☒	☐	For hierarchical and complex designs, identification of the appropriate level for tests and full reporting of outcomes
☐	☒	Estimates of effect sizes (e.g. Cohen's *d*, Pearson's *r*), indicating how they were calculated
☒	☐	Clearly defined error bars *State explicitly what error bars represent (e.g. SD, SE, CI)*

Our web collection on statistics for biologists may be useful.

Software and code

Policy information about availability of computer code

Data collection	The scripts to download and process the datasets are available at the dynbenchmark repository: https://www.github.com/dynverse/dynbenchmark
Data analysis	The data analysis was conducted using several custom software package, all available at https://www.github.com/dynverse/dynverse

For manuscripts utilizing custom algorithms or software that are central to the research but not yet described in published literature, software must be made available to editors/reviewers upon request. We strongly encourage code deposition in a community repository (e.g. GitHub). See the Nature Research guidelines for submitting code & software for further information.

Data

Policy information about availability of data

All manuscripts must include a data availability statement. This statement should provide the following information, where applicable:

- Accession codes, unique identifiers, or web links for publicly available datasets
- A list of figures that have associated raw data
- A description of any restrictions on data availability

Data is deposited in Zenodo (https://zenodo.org/record/1443566) with doi 10.5281/zenodo.1443566

Field-specific reporting

Please select the best fit for your research. If you are not sure, read the appropriate sections before making your selection.

☒ Life sciences ☐ Behavioural & social sciences ☐ Ecological, evolutionary & environmental sciences

For a reference copy of the document with all sections, see nature.com/authors/policies/ReportingSummary-flat.pdf

Life sciences study design

All studies must disclose on these points even when the disclosure is negative.

Sample size	In our case the number of datasets used to compare methods corresponds to the sample size. We included as many real datasets as we could find.
Data exclusions	No data was excluded from the study
Replication	To make sure that the experimental findings are reproducible, we (1) verified that the performance of the methods is similar between different dataset sources, (2) verified that the metrics we use produce the expected results under a varied set of perturbational settings and (3) assessed the stability of each method.
Randomization	This is not relevant to our study because we do not include separate experimental groups
Blinding	This is not relevant to our study because we do not include separate experimental groups

Reporting for specific materials, systems and methods

Materials & experimental systems

n/a	Involved in the study
☒	☐ Unique biological materials
☒	☐ Antibodies
☒	☐ Eukaryotic cell lines
☒	☐ Palaeontology
☒	☐ Animals and other organisms
☒	☐ Human research participants

Methods

n/a	Involved in the study
☒	☐ ChIP-seq
☒	☐ Flow cytometry
☒	☐ MRI-based neuroimaging

nature research

Corresponding author(s): Yvan Saeys

Code and Software Submission Checklist

Prior to submitting your work to Nature Research, we strongly recommend that you ask at least one colleague who is unfamiliar with your software to install the tool(s), follow the instructions, and provide feedback. This process will help ensure that reviewers will also be able to run your software.

You must submit all required content as a single zip file prior to peer review or provide a link where editors and reviewers can access all required content.

Required content

- [x] Compiled standalone software and/or source code
- [x] A small (simulated or real) dataset to demo the software/code

A README file that includes:

1. System requirements
   - [x] All software dependencies and operating systems (including version numbers)
   - [x] Versions the software has been tested on
   - [x] Any required non-standard hardware
2. Installation guide
   - [x] Instructions
   - [x] Typical install time on a "normal" desktop computer
3. Demo
   - [x] Instructions to run on data
   - [x] Expected output
   - [x] Expected run time for demo on a "normal" desktop computer
4. Instructions for use
   - [x] How to run the software on your data
   - [x] (OPTIONAL) Reproduction instructions

   We encourage you to include instructions for reproducing all the quantitative results in the manuscript.

Additional information

Describe your software's license for use. We strongly recommend using a license approved by the Open Source Initiative.

GPL-3

Provide a link to the code in an open source repository (when available).

https://github.com/dynverse/dynbenchmark and https://github.com/dynverse/dynguidelines

Your manuscript should include a complete, detailed description of the code's functionality (i.e. pseudocode).

Please indicate where this is found:

- [ ] Main text
- [ ] Methods section
- [x] Elsewhere (specify):

The descriptions of the code can be found in the READMEs in the github repository

Examples of well-structured software packages

1. https://github.com/neurodata-papers/MGC
2. https://github.com/neurodata-papers/LOL
3. https://www.nature.com/nbt/journal/v34/n6/abs/nbt.3569.html#supplementary-information
4. https://www.nature.com/nature/journal/v548/n7669/full/nature23463.html#extended-data
   https://github.com/yasharhezaveh/Ensai
5. https://www.nature.com/nbt/journal/v34/n11/full/nbt.3685.html#supplementary-information
   https://github.com/IFIproteomics/LFQbench