CellRank for directed single-cell fate mapping

Marius Lange
Institute of Computational Biology, Helmholtz Center Munich https://orcid.org/0000-0002-4846-1266
Volker Bergen
Institute of Computational Biology, Helmholtz Center Munich
Michal Klein
Institute of Computational Biology, Helmholtz Center Munich
Manu Setty
Sloan Kettering Institute, Memorial Sloan Kettering Cancer Center
Bernhard Reuter
Department of Computer Science, University of Tübingen
Mostafa Bakhti
Institute of Diabetes and Regeneration Research, Helmholtz Center Munich,
Heiko Lickert
Helmholtz Zentrum München https://orcid.org/0000-0002-4597-8825
Meshal Ansari
Institute of Computational Biology, Helmholtz Center Munich
Janine Schniering
Institute of Lung Biology and Disease (ILBD), Helmholtz Zentrum München
Herbert Schiller
Helmholtz Zentrum München, Institute of Lung Biology and Disease, Group Systems Medicine of Chronic Lung Disease, Member of the German Center for Lung Research (DZL), CPC-M bioArchive,Munich https://orcid.org/0000-0001-9498-7034
Dana Pe'er
Memorial Sloan Kettering Cancer Center https://orcid.org/0000-0002-9259-8817
Fabian Theis (fabian.theis@helmholtz-muenchen.de)
Helmholtz Zentrum München https://orcid.org/0000-0002-2419-1943

Article

Keywords: CellRank, directed, single-cell fate mapping

Posted Date: October 29th, 2020

DOI: https://doi.org/10.21203/rs.3.rs-94819/v1

Version of Record: A version of this preprint was published at Nature Methods on January 13th, 2022.
See the published version at https://doi.org/10.1038/s41592-021-01346-6.

October 19, 2020

CellRank - Online Methods

Marius Lange[1,2], Volker Bergen[1,2], Michal Klein[1], Manu Setty[3], Bernhard Reuter[4,5], Mostafa Bakhti[6,7], Heiko Lickert[6,7], Meshal Ansari[1,8], Janine Schniering[8], Herbert B. Schiller[8], Dana Pe'er[3*], Fabian J. Theis[1,2,9*]

1 Institute of Computational Biology, Helmholtz Center Munich, Germany.
2 Department of Mathematics, TU Munich, Germany.
3 Program for Computational and Systems Biology, Sloan Kettering Institute, Memorial Sloan Kettering Cancer Center, New York, NY, USA.
4 Department of Computer Science, University of Tübingen, Germany.
5 Zuse Institute Berlin (ZIB), Takustr. 7, 14195 Berlin, Germany.
6 Institute of Diabetes and Regeneration Research, Helmholtz Center Munich, Germany.
7 German Center for Diabetes Research (DZD), Neuherberg, Germany.
8 Comprehensive Pneumology Center (CPC) / Institute of Lung Biology and Disease (ILBD), Helmholtz Zentrum München, Member of the German Center for Lung Research (DZL), Munich, Germany
9 TUM School of Life Sciences Weihenstephan, Technical University of Munich, Germany.
*Corresponding authors: fabian.theis@helmholtz-muenchen.de and peerd@mskcc.org

Contents

1 The CellRank algorithm . . . 2
1.1 Modelling approach . . . 3
1.2 Computing the transition matrix . . . 5
1.3 Coarse-graining the Markov chain . . . 7
1.4 Computing fate probabilities . . . 10
1.5 Propagating velocity uncertainty . . . 12
1.6 The CellRank software package . . . 14
2 Computing a directed PAGA graph . . . 16
3 Computing gene expression trends along lineages . . . 16
4 Clustering gene expression trends . . . 17
5 Uncovering putative driver genes . . . 18
6 Robustness analysis . . . 18
7 Pancreas data example . . . 19
8 Lung data example . . . 20
9 Methods comparison . . . 21
10 Immunofluorescence stainings and microscopy on airway epithelial cells . . . 23

Online methods

1 The CellRank algorithm

The aim of the CellRank algorithm is to detect the initial, terminal and intermediate states of a cellular system and to define a global map of fate potentials that assigns each cell to these states in a probabilistic manner. Given our inferred fate potentials, we compute gene expression trends along trajectories in the fate map and provide several possibilities for visualizing these. The inputs to CellRank are a count matrix $X \in \mathbb{R}^{N \times G}$ where N is the number of cells and G is the number of genes as well as a velocity matrix $V = \mathbb{R}^{N \times G}$, defining a vector field representing RNA velocity[1,2] for each cell and gene. Note that CellRank can be generalized to any kind of vector field, i.e. V could equally represent directed information given by e.g. metabolic labeling[3–6]. There are three main steps to the CellRank algorithm:

1. Compute transition probabilities among observed cells. These reflect how likely a cell with a given cell state, defined by its gene expression profile, is to change its profile to that of a target cell. We compute these probabilities by integrating two sources of evidence: (1) transcriptomic similarity between the source and target cells and (2) an extrapolation of a cell's current gene expression profile into the near future using RNA velocity. We aggregate these transition probabilities in the transition matrix P and use it to model cell-state transitions as a Markov chain.

2. Coarse-grain the Markov chain into a set of initial, terminal and intermediate macrostates of cellular dynamics. Each cell is assigned to each macrostate via a membership matrix χ. The assignment is soft, i.e. each cell has a certain degree of confidence of belonging to each macrostate. We compute transition probabilities among macrostates in the matrix P_c. This matrix allows us to identify whether macrostates are initial, terminal or intermediate.

3. Compute fate probabilities towards a subset of the macrostates. This will typically include the terminal states, but can also include intermediate states, depending on the biological question. We compute how likely each cell is to transition into each of the selected macrostates and return these probabilities in a fate matrix F.

CellRank extracts the essence of cellular state transitions The principle of the CellRank algorithm is to decompose the dynamics of the biological system into a set of dynamical macrostates. We target macrostates that are associated with regions in the phenotypic manifold which cells are unlikely to leave once they have entered them. For each observed cell, we compute how likely it is to belong to each of these macrostates. We accumulate these soft assignments in a membership matrix $\chi \in \mathbb{R}^{N \times n_s}$. Further, we compute a coarse-grained transition matrix $P_c \in \mathbb{R}^{n_s \times n_s}$ which specifies transition probabilities among macrostates. The coarse-grained transition matrix allows us to reduce the biological system to its essence: dynamical macrostates of observed cell-state transitions and their relationship to one another. Based on the coarse-grained transition matrix, we classify macrostates as either initial, intermediate or terminal. Initial states will be macrostates that have very small incoming but large outgoing transition probability. Intermediate states will be macrostates that have both incoming and outgoing transition probability. Terminal states will be macrostates that have large incoming but very little outgoing and large self-transition probability.

CellRank computes probabilistic fate potentials Each macrostate is associated with a subset of the observed cells via the membership matrix χ. Once we classified macrostates as either initial, intermediate or terminal using the coarse-grained transition matrix P_c, we may ask how likely each cell is to transition to each of the n_t terminal states. CellRank efficiently computes these probabilities and returns a fate matrix $F \in \mathbb{R}^{N \times n_t}$. The matrix F extends the short-range fate relationships given by RNA velocity to the global scale: from initial to terminal states along the phenotypic manifold.

We account for high noise levels in the velocity vectors via a stochastic Markov chain formulation, by restricting predicted transitions to align with the phenotypic manifold and by propagating velocity uncertainty into the Markov chain.

CellRank uncovers gene expression trends towards specific terminal populations The outputs of the CellRank algorithm are

- a membership matrix $\chi \in \mathbb{R}^{N \times n_s}$ where n_s is the number of macrostates. Row i in χ softly assigns cell i to any of the macrostates.
- a coarse-grained transition matrix $P_c \in \mathbb{R}^{n_s \times n_s}$ that describes how likely these macrostates are to transition into one another. The matrix P_c allows macrostates to be classified as either initial, intermediate or terminal.
- a fate matrix $F \in \mathbb{R}^{N \times n_t}$ where n_t is the number of terminal states. Row i in F specifies how likely cell i is to transition towards any of the terminal states.

We use the fate matrix F to model gradual lineage commitment. Fate biases can be aggregated to the cluster level and visualized as pie charts on a new directed version of PAGA graphs[7] (Section 2). Further, we use the fate matrix F to uncover gene expression trends towards the identified terminal states (Section 3). Once the trends have been fit, they can be clustered to discover the main regulatory dynamics towards different terminal states (Section 4). For the identification of putative regulators towards specific terminal states, we correlate gene expression values with fate probabilities (Section 5).

1.1 Modelling approach

Similarly to other methods[8–10], CellRank models cell state transitions among observed cellular profiles. Unlike other velocity based methods, following the success of pseudotime methods, key to our model is that we restrict possible state changes to those consistent with the global structure of the phenotypic manifold via a KNN graph computed based on similarities in gene expression space. Our approach then biases the likely future state of an observed cell within its local graph neighborhood based on RNA velocity, by combining transcriptional similarity with RNA velocity to direct edges in the graph and to assign a probability to each cell state transition. When computing these probabilities, we take into account uncertainty in the velocity vectors. By aggregating individual, stochastic transitions within the global structure of the phenotypic manifold, we uncover the fate bias for individual cells. We make the following assumptions:

- state transitions are gradual, daughter cells are in general transcriptomically similar to their mother cells. Cells traverse a low-dimensional phenotypic manifold from initial to terminal states via a set of intermediate states.
- the set of sampled cellular profiles spans the entire state change trajectory, i.e. intermediate states have been covered, there are no 'gaps' in the trajectory.
- while for an individual cell, its past history is stored in epigenetic modifications, we model average cellular dynamics where state transitions occur without memory.
- RNA velocity approximates the first derivative of gene expression. This must not precisely hold for every gene in each individual cell as we treat state transitions as a stochastic process, enforce alignment with the manifold and propagate uncertainty, but it should hold in expectation for enough cells so that we are able to estimate the overall directional flow.

Based on these assumptions, we model cellular state transitions using a Markov chain: a stochastic process $X = (X_t)_{t \in T}$ – a sequence of random variables $X_t : \Omega \to E$ on a probability space $(\Omega, \mathcal{A}, \mathbb{P})$ over a countable set Ω mapping to a measurable state space (E, Σ) – that describes the evolution

of a probability distribution over time where the future distribution only depends on the current distribution and not on the past, i.e. $\Pr(X_{t_{n+1}} = x \mid X_{t_1} = x_1, X_{t_2} = x_2, ..., X_{t_n} = x_n) = \Pr(X_{t_{n+1}} = x \mid X_{t_n} = x_n)$. We use a Markov chain over a discrete and finite state space Ω, where each state in the chain is given by an observed cellular transcriptional profile. To define the Markov chain, we need to compute a transition matrix $P \in \mathbb{R}^{N \times N}$ which describes how likely one cell is to transition into another. We construct $P \in \mathbb{R}^{N \times N}$ using a KNN graph based on transcriptional similarity between cells and a given vector field. While CellRank generalizes to any given vector field, we demonstrate it using RNA-velocities, based on unspliced to spliced read ratios, computed with scVelo[9].

Defining initial, intermediate and terminal states in biological terms We define an initial (terminal) state as an ensemble of measured gene expression profiles which, when taken together, characterize the starting (end) point of one particular cell-state change. We define an intermediate state as an ensemble of gene expression profiles which, when taken together, characterize a point on the cell-state transition trajectory which lies in between one or several initial and terminal states.

Translating initial, intermediate and terminal states into mathematical terms To translate the above terms into mathematics, we make use of the coarse-graining given by the membership matrix χ and the coarse-grained transition matrix P_c. We show below that our assignment of cells to macrostates maximizes a criterion we call the *crispness*: we obtain macrostates which have little overlap and large self-transition probability. In other words: we recover the kinetics of the Markov chain on slow-time scales, i.e. macrostates and their transitions reflect the limiting behavior of the Markov chain. Among the set of macrostates, we identify initial states as those which have little incoming large but large outgoing transition probability in P_c. Intermediate states will have both incoming and outgoing transition probability in P_c. Terminal states will have large incoming but little outgoing and large self-transition probability in P_c. An important term in the mathematical framework is *metastability*: a process starting in a metastable state will stay there with high probability for a long time. Accordingly, we define a metastable state of cellular dynamics as an area in phenotypic space that cells are unlikely to leave again once they have entered. A metastable state will typically correspond to a terminal state, while an intermediate state is typically only weakly metastable. Initial states can constitute weakly metastable states, if the probability of leaving them is small, potentially because of heavily cycling populations.

Reversing the Markov chain to recover initial states Initial states may not be picked up as macrostates during coarse-graining of the Markov chain because they are not stable enough, i.e. cells in the initial state have very little probability of transitioning into one another and rapidly start traversing their state change trajectory. In these cases, we reverse the Markov chain, i.e. we flip the arrows in the velocity vector field V. The initial state now constitutes a terminal (i.e. metastable) state of the reversed dynamics and may be recovered by coarse-graining and interpreting the reversed Markov chain.

Defining fate probabilities towards macrostates Biologically, we define the fate probability of cell i to reach macrostate $j \in 1, ..., M$ as the probability of cell i executing a series of gene expression programs which change its phenotype to match the phenotype of cells in macrostate j. Within the context of fate probabilities, we will typically be interested in macrostates which are either terminal or intermediate states. Mathematically, we translate this to the probability of a random walk on the Markov chain initialized in cell i to reach any cell belonging to macrostate j before reaching any cell belonging to another macrostate. CellRank efficiently computes these probabilities in closed form using absorption probabilities (Subsection 1.4).

1.2 Computing the transition matrix

We model each observed cell by one microstate in the Markov chain. To compute transition probabilities among cells, we make use of transcriptomic similarity to define the global topology of the phenotypic manifold and of RNA velocity to direct local movement on the manifold. To model the global topology of the phenotypic manifold, the first step of the CellRank algorithm is to compute a KNN graph.

Computing a KNN graph to align local transitions with global topology We compute a KNN graph to constrain the set of possible transitions to those that are consistent with the global topology of the phenotypic manifold: Each cell is thus only allowed to transition into one of its K nearest neighbors. While CellRank can generalize to any reasonable similarity kernel, here, we compute the KNN graph as follows:

1. project the data onto the first L principal components to obtain a matrix $X_{PC} \in \mathbb{R}^{N \times L}$, where rows correspond to cells and columns correspond to PC features.
2. for each cell i, compute distances to its K-nearest neighbors based on euclidean distance in X_{PC}. Accumulate distances in a matrix $D \in \mathbb{R}^{N \times N}$.
3. the KNN relationship will lead to a directed graph because it is not a symmetric relationship. Symmetrize the KNN relations encoded by D, such that cells i and j are nearest neighbors if either i is a nearest neighbors of j, or j is a nearest neighbors of i. This will yield an undirected symmetric version D_{sym} of D, where each cell has at least K nearest neighbors.
4. compute a symmetric adjacency matrix A based on D_{sym} containing similarity estimates between neighboring cells according to the manifold structure. To approximate cell similarities, we use the method implemented in the UMAP algorithm, which adapts the singular set and geometric realization functors from algebraic topology to work in the context of metric spaces and fuzzy simplicical sets[11,12].

We choose $K = 30$ to be the number of nearest neighbors by default. We show in Supplementary Fig. 8a that CellRank is robust to the choice of K. To compute the similarity metric, the option presented is the default in SCANPY[13]. Alternatively, similarity may be computed using a Gaussian kernel with density-scaled kernel width as introduced by ref.[14] and adapted to the single cell context in ref.[10]. We choose $L = 30$ to be the number of principal components by default. This can be adapted based on knee-point heuristics or the percentage of variance explained, however, we show in Supplementary Fig. 8d that CellRank is robust to the exact choice of L.

Directing the KNN graph based on RNA Velocity Next, we direct the edges of the KNN graph using RNA velocity information, giving higher probability to those neighbors whose direction best aligns with the direction of the velocity vector. Specifically, for cell i with gene expression profile $x_i \in \mathbb{R}^G$ and velocity vector $v_i \in \mathbb{R}^G$, consider its neighbors $j = 1, 2, ..., K_i$ with gene expression profiles $\{x_1, x_2, ..., x_K\}$. Note that the graph construction outlined above leads to a symmetric KNN graph, where K_i is not constant across all cells, but $K_i \geq K \, \forall i \in \{1, ..., N\}$. For each neighboring cell k, compute the corresponding state-change vector with cell i, $s_{ik} = x_k - x_i \in R^G$. Next, we compute Pearson correlations $c_i \in R^K$ of v_i with all state change vectors via

$$c_{ik} = \frac{(s_{ik} - e\bar{s}_{ik})^\top (v_i - e\bar{v}_i)}{\|s_{ik} - e\bar{s}_{ik}\| \, \|v_i - e\bar{v}_i\|} \in [-1, 1]^K \,, \tag{1}$$

where e is a constant vector of ones and $\bar{s}_{ik}$ and $\bar{v}_i$ are averages over the state change vector and the velocity vector, respectively. Intuitively, c_i contains the cosines of the angles that the mean-centered v_i forms with the mean-centered state-change vectors s_{ik}. A value of 1 means perfect correlation between the gene expression changes predicted by the local velocity vector and the actual change

observed when going from the reference cell to any of its nearest neighbors. Pearson correlations have been computed in similar ways by scvelo[9] and velocyto[1] to project the velocity vectors into a given embedding. In Subsection 1.5 below, we show how their ideas can be formalized and extended to account for uncertainty in the velocity vector.

Transforming correlations into transition probabilities To use the vector c_i as a set of transition probabilities to neighboring cells, we need to make sure it is positive and sums to one. For cell i, define a set of transition probabilities $p_i \in \mathbb{R}^K$ via

$$p_{ik} = \frac{\exp(\sigma c_{ik})}{\sum_{l=1}^{K} \exp(\sigma c_{il})} \tag{2}$$

where $\sigma > 0$ is a scalar constant that controls how centered the categorical distribution will be around the most likely value, i.e. around the state-change transition with maximum correlation (see below). We repeat this for all (i, k) which are nearest neighbors to compute the transition matrix $P_v \in \mathbb{R}^{N \times N}$. This scales linearly in the number of cells N, the number of nearest neighbors K and the number of genes G as the KNN graph is sparse.

Automatically determine σ We reasoned that the value of σ should depend on typical Pearson correlation's between velocity vectors and state change vectors observed in the given data-set. For this reason, we use the following heuristic:

$$\sigma = \frac{1}{\text{median}(\{|c_{ik}| \; \forall i, k\})} \, . \tag{3}$$

This means that if the median absolute Pearson correlation observed in the data is large (small), we use a small (large) value for σ. The intuition behind this is that for sparsely sampled data-sets where velocity vectors only roughly point into the direction of neighboring cells, we upscale all correlations a bit. Typical values for σ we compute this way range from 1.5 (lung example[15]) to 3.8 (pancreas example[16]).

Coping with uncertainty in the velocity vectors scRNA-seq data is a noisy measurement of the underlying gene expression state of individual cells. RNA velocity is computed on the basis of these noisy measurements and is therefore itself a substantially noisy quantity. In particular the unspliced reads required by velocity and scVelo to estimate velocities are very sparse and their abundance varies depending on the amount of relevant intronic sequence of different genes. Besides this inherent noise, preprocessing decisions in the alignment pipeline of spliced and unspliced reads have been shown to impact the final velocity estimate[17]. Further uncertainty in the velocity estimate arises because assumptions have to be made which may not always be satisfied in practice:

- the original velocyto[1] model assumes that for each gene, a steady state is captured in the data. The scVelo[9] model circumvents this assumption by dynamic modeling, extending RNA velocity to transient cell populations, however there is often only a sparsity of transitional cells to estimate these dynamics.
- both models assume that the key biological driver genes for a given cell-state transition are intron rich and may therefore be used to estimate spliced to unspliced ratios. This has been shown to be the case in many neurological settings, however, in other systems such as hematopoeisis, it remains unclear whether this assumption is met.
- both models assume that per gene, a single set of kinetic parameters α (transcription rate), β (splicing rate) and γ (degratation rate) may be used across all cells. However, we know that in many settings, this assumption is violated because of alternative splicing or cell-type specific regulation[18–21].

- both models assume that there are no batch effects present in the data. To date, to the best of our knowledge, there are no computational tools to correct for batch effects in velocity estimates.
- both models assume that the cell-state transition captured in the data is compatible with the time scale of splicing kinetics. However, this is often not known a priori and may explain the limited success of RNA velocity in studying hematopoeisis to date.

The points outlined above highlight that RNA velocity is a noisy, uncertain estimate of the likely direction of the future cell state. To cope with the uncertainty present in RNA velocity, we adapt four strategies:

- we restrict the set of possible transitions to those consistent with the global topology of the phenotypic manifold as described by the KNN graph.
- we use a stochastic formulation based on Markov chains to describe cell-state transitions. For cell i with velocity vector v_i, we allow transitions to each nearest neighbor j with transition probability p_{ij}. This means that we even allow transitions backwards, against the flow prescribed by the velocity vector field, with small probability. This reflects our uncertainty in v_i.
- we combine RNA velocity information with trancriptomic similarity, see below.
- we propagate uncertainty in v_i into the downstream computations (Subsection 1.5).

Emphasizing transcriptomic similarity Thus far, we have combined RNA velocity with transcriptomic similarity by computing a similarity-based KNN graph to restrict the set of possible transitions. To further take advantage of the information captured by the KNN graph and to increase robustness of the algorithm with respect to noisy velocity vectors, we combine the velocity based transition matrix P_v with a similarity based transition matrix P_s via

$$P = \lambda P_v + (1 - \lambda) P_s \quad \text{for } \lambda \in [0, 1]\,. \tag{4}$$

The matrix P_s is computed by row-normalizing the adjacency matrix A. In practical applications, we have found that using values around $\lambda = 0.2$ increase robustness with respect to noisy velocity estimates. The matrix P is the final transition matrix estimated by the CellRank algorithm.

1.3 Coarse-graining the Markov chain

The transition matrix P defines a Markov chain among the set of all observed cells, where each cell constitutes a microstate of the Markov chain. However, it is difficult to directly use P to interpret the cellular trajectory because P is a fine-grained, noisy representation of cell state transitions. Therefore, we seek to reduce P to its essence: macrostates representing key biological states and their transition probabilities among each other. We accomplish this using the *Generalized Perron Cluster Cluster Analysis* (GPCCA)[22,23], a method originally developed to study conformational dynamics in proteins. We adapt it to the single cell setting and utilize it to project the large transition matrix P onto a much smaller coarse-grained transition matrix P_c that describes transitions among a set of macrostates. A macrostate is associated with a subset M of the state space $M \subset \Omega$. The macrostates are defined through a so-called membership matrix χ. Rows of χ contain the soft assignment of cells to macrostates.

Generalized Perron Cluster Cluster Analysis (GPCCA) The aim of the GPCCA algorithm is to project the large transition matrix P onto a much smaller coarse-grained transition matrix P_c, which describes transitions between macrostates of the biological system[22,23]. For the projected or

embedded dynamics to be Markovian, we require the projection to be based on an invariant subspace of P (invariant subspace projection), i.e. a subspace W for which

$$P^\top x \in W \ \forall x \in W\,. \tag{5}$$

In case of a reversible P, invariant subspaces are spanned by the eigenvectors of P[24]. In our case however, P is non-reversible and the eigenvectors will in general be complex. Since the GPCCA algorithm can not cope with complex vectors, we rely on real invariant subspaces of the matrix P for the projection. Such subspaces are provided by the real Schur decomposition of P[22,23],

$$P = QRQ^\top\,, \tag{6}$$

where $Q \in \mathbb{R}^{N\times N}$ is orthogonal and $R \in \mathbb{R}^{N\times N}$ is quasi-upper triangular[25]. R has 1-by-1 or 2-by-2 blocks on the diagonal, where the former are given by the real eigenvalues and the latter are associated with pairs of complex conjugate eigenvalues.

Invariant subspaces of the transition matrix Columns of Q corresponding to real eigenvalues span real invariant subspaces. Columns of Q corresponding to pairs of complex conjugate eigenvalues span real invariant subspaces when kept together, but not if they are separated. Particularly, for columns q_j and q_k of Q belonging to a pair of complex conjugate eigenvalues, the space $W_0 = \mathrm{span}(q_j, q_k)$ is invariant under P, but the individual q_j and q_k are not[26]. Depending on the constructed subspace, different dynamical properties of P will be projected onto P_c. Choosing Schur vectors belonging to real eigenvalues close to 1, metastabilities are recovered, while for Schur vectors with complex eigenvalues close to the unit circle, cyclic dynamics are recovered[22,23]. Both options are available in CellRank, defaulting to the recovery of metastabilities.

Projecting the transition matrix Let $\bar{Q} \in \mathbb{R}^{N\times n_s}$ be the matrix formed by selecting n_s columns from Q according to some criterion (metastability or cyclicity). Let $\chi \in \mathbb{R}^{N\times n_s}$ be a matrix obtained via linear combinations of the columns in $\bar{Q}$, i.e.

$$\chi = \bar{Q}A\,, \tag{7}$$

for an invertible matrix $A \in \mathbb{R}^{n_s\times n_s}$. We obtain the projected transition matrix via a Galerkin projection[22,23],

$$P_c = (\chi^\top D\chi)^{-1}(\chi^\top DP\chi)\,. \tag{8}$$

Here, the matrix D is the diagonal matrix of a weighted scalar product. The Schur vectors must be orthogonal with respect to this weighted scalar product, i.e. $Q^\top DQ = I$ with the n_s-dimensional unit matrix I, to yield the required invariant subspace projection. The diagonal elements of D are in principle arbitrary, but a convenient choice would be the uniform distribution or some distribution of the cellular states of interest. Choosing the uniform distribution, as is the default in CellRank, would result in a indiscriminate handling (without imposing any presumptions about their distribution) of the cellular states. Note that the matrix inversion in Equation (8) is performed on a very small matrix of size $n_s \times n_s$.

Computing the membership vectors In principle, we could use any invertible A in Equation (7). However, we would like to interpret the rows of of χ as membership vectors that assign cells to macrostates. For this reason, we seek a matrix A that minimizes the overlap between the membership vectors χ, i.e. a matrix A that minimizes off-diagonal entries in $\chi^\top D\chi$. This is equivalent to maximizing

$$\mathrm{trace}(\tilde{D}^{-1}\chi^\top D\chi)\,, \tag{9}$$

where $\tilde{D}$ is chosen to row-normalize $S = \tilde{D}^{-1}\chi^\top D\chi$,

$$\tilde{D}^{-1} = \mathrm{diag}\left(\frac{1}{\sum_j(\chi^\top D\chi)_{1j}}, ..., \frac{1}{\sum_j(\chi^\top D\chi)_{n_s j}}\right) . \tag{10}$$

Choosing Schur vectors with real eigenvalues close to one, thus recovering metastability, maximizing Equation (9) can be interpreted as maximizing the metastability of the macrostates in the system. In practice, we use

$$f_{n_s}(A) = n_s - \mathrm{trace}(S)\,, \tag{11}$$

as our objective function, which is bounded below by zero and convex on the feasible set defined through the linear constraints[24]. We must minimize f_{n_s} with respect to the constraints

$$\sum_j \chi_{ij} = 1 \quad \forall i \in \{1, ..., N\} \quad \text{(partition of unity)} \quad , \tag{12}$$

$$\chi_{ij} \geq 0 \quad \forall i \in \{1, ..., N\},\, j \in \{1, ..., n_s\} \quad \text{(positivity)} \quad , \tag{13}$$

which is not a trivial task. Among the several possibilities to solve the minimization problem, a convenient choice is to perform unconstrained optimization on $A(2 : n_s, 2 : n_s)$ using a trick: to impose the constraints after each iteration step, thus transforming the (unfeasible) solution into a feasible solution[24]. The drawback of this approach is that this routine is non-differentiable. Thus a derivative-free method like the Nelder-Mead method, as implemented in the Scipy routine `scipy.optimize.fmin`, should be used for the optimization.

Positivity of the projected transition matrix Note that the projected transition matrix P_c may have negative elements if macrostates share a large overlap. In practice, this is caused by a suboptimal number of macrostates n_s and can be resolved by changing that number. We may interpret P_c as the transition matrix of a Markov chain between the set of macrostates if it is non-negative within numerical precision.

Tuning the number of macrostates The number of macrostates n_s can be chosen in a number of different ways:

- using the eigengap heuristic for the real part of the eigenvalues close to one.
- define the crispness ξ of the solution as the value of $\mathrm{trace}(\tilde{D}^{-1}\chi^\top D\chi)/n_s$. The larger this value, the smaller the overlap between the macrostates, and in turn, the sharper or "crisper" the recovered macrostates. The crispness can be computed for different numbers of macrostates n_s and the number n_s with the largest value of ξ should be selected.
- to avoid having to solve the full problem for too many values of n_s, do a pre-selection using the minChi criterion: Based on an initial guess for A, compute a membership matrix χ and calculate $\mathrm{minChi} = \min_{i,j}(\chi_{ij})$. In general, this value will be negative because the starting guess is infeasible. The closer to zero the value of minChi is, the more we can expect n_s to yield a crisp decomposition of the dynamics.
- combining the minChi criterion and the crispness, to avoid solving the full problem for many n_s, but still select the n_s with the crispest decomposition. This is done by first selecting an interval of potentially good numbers of macrostates n_s via the minChi criterion and afterwards using the crispness to select the best n_s from the preselected macrostate numbers.

All of the above are available through CellRank.

Scalable Python implementation of GPCCA Following the original MATLAB implementation[22,23], we wrote up GPCCA as a general algorithm in Python and included it in the `MSMTools`[27] package, which is widely used for studying protein folding kinetics. From CellRank, we interface to `MSMTools` for the GPCCA algorithm. A naive implementation of the Schur decomposition would scale cubically in cell number. We alleviate this problem by using `SLEPSc` to compute a partial real Schur decomposition using an iterative, Krylov-subspace based algorithm that optimally exploits the sparsity structure of the transition matrix[28,29]. Overall, this reduces the computational complexity of our algorithm to be almost linear in cell number (Fig. 5d and Supplementary Table 1). This allows CellRank to scale well to very large cell numbers.

Automatically determine terminal states We use the coarse-grained dynamics given by P_c to automatically identify terminal states. The idea is to look for the most stable macrostates according to the coarse-grained transition matrix P_c. Define the *stability index* (SI) of a macrostate $m \in \{1, ..., n_s\}$ through its corresponding diagonal value in P_c, i.e. through its self-transition probability $p_{c\,(m,m)}$. The intuition behind this is that cells in terminal populations should have very little probability to transition to cells in other populations and should distribute most (if not all) of their probability mass to cells from the same terminal population. To identify the number of terminal states, we set a threshold on SI, i.e. we classify all states as terminal for which SI $\geq \epsilon_{\text{SI}}$ with $\epsilon_{\text{SI}} = 0.96$ by default.

Automatically determine initial states To identify the initial states automatically, we introduce the *coarse-grained stationary distribution* (CGSD) π_p, given by

$$\pi_p = \chi^\top \pi \tag{14}$$

where π is the stationary distribution of the original transition matrix P. The stationary distribution satisfies

$$\pi^\top P = \pi^\top,\ \pi_i > 0\ \forall i \text{ and } \sum_i \pi_i = 1. \tag{15}$$

In other words, the stationary distribution π is an invariant measure of P and can be computed by normalizing the top left eigenvector of P (corresponding to the eigenvalue 1). Under certain conditions (ergodicity, see ref.[30]) imposed on the Markov chain, the stationary distribution is the distribution that the process converges to, if it evolves long enough, i.e. it describes the long-term evolution of the Markov chain. In the same vein, the CGSD π_c describes the long-term evolution of the Markov chain given by the coarse-grained transition matrix P_c. The CGSD π_c assigns large (small) values to macrostates that the process spends a large (little) amount of time in, if it is run infinitely long. As such, we may use it to identify initial states by looking for macrostates which are assigned the smallest values in π_c. The intuition behind this is that initial states should be states that the process is unlikely to visit again once it has left them. The number of initial states is a method parameter set to one by default which can be modified by the user to detect several initial states.

Automatically determine intermediate states All remaining macrostates, i.e. macrostates which have neither been classified as terminal nor as initial, are classified as intermediate. Biologically, these correspond to intermediate, transient cell populations on the state change trajectory.

1.4 Computing fate probabilities

Given the soft assignment of cells to macrostates by χ and the identification of terminal states through P_c, we compute how likely each cell is to transition towards these terminal states. Let n_t

be the number of terminal states. For the sake of clarity, we assume that we are only interested in fate probabilities towards terminal states, however, the below computations apply just as well to intermediate states, depending on the biological question. For each terminal macrostate t for $t \in \{1, ..., n_t\}$, we choose f cells which are strongly assigned to t according to χ. That is, for terminal macrostate t, we extract the corresponding column from χ and we calculate the terminal index set $\mathcal{R}_t$ of cells which have the largest values in this column of χ. If cell i is part of terminal index set $\mathcal{R}_t$, we assume cell i is among the f most eligible cells to characterize the terminal macrostate t in terms of gene expression. We store the indices of the remaining cells in the transient index set $\mathcal{T}$. The index sets $\{\mathcal{R}_t \,|\, t \in \{1, ..., n_t\}\}$ and $\mathcal{T}$ form a disjoint partition of the state space, which means they do not overlap and they cover the entire state space. For each cell i in $\mathcal{T}$, we would like to compute a vector of probabilities $f_i \in \mathbb{R}^{n_t}$ which specifies how likely this cell is to transition into any of the terminal sets $\mathcal{R}_t$. To interpret f_i as a categorical distribution over cell fate, we require $f_{i,t} \geq 0 \; \forall i \in \mathcal{T} \; \forall t \in \{1, ..., n_t\}$ and $\sum_t f_{i,t} = 1 \; \forall i \in \mathcal{T}$. We accumulate the f_i column-wise in the fate matrix $F \in R^{N \times n_t}$.

Absorption probabilities reveal cell fates We could approximate the f_i based on sampling: initialise a random walk on the Markov chain in cell i. Continue to simulate the random walk until any cell from a terminal set $\mathcal{R}_t$ is reached. Record t and repeat this many times. Finally, count how often random walks initialized in cell i terminated in any of the terminal index sets $\mathcal{R}_t$. In the limit of repeating this infinitely many times, the normalized frequencies over reaching either terminal set will be equal to the desired fate probabilities for cell i, under reasonable assumptions on the Markov chain (irreducibility). Luckily, we do not have to do this in a sampling based approach, we can exploit the fact that a closed form solution exists for this problem: absorption probabilities.

Computing absorption probabilities Key to the concept of absorption probabilities are *recurrent* and *transient* classes, which we will define here for the present case of a finite and discrete state space. Let $i \in \Omega$ and $j \in \Omega$ be two states of the Markov chain. In our case, i and j are cells. We say that i is *accessible* from j, if and only if there exists a path from j to i according to the transition matrix P. A *path* is a sequence of transitions which has non-zero transition probability. Further, i and j *communicate*, if and only if i is accessible from j and j is also accessible from i. Communication defines an equivalence relation on the state space Ω, i.e. it is a reflexive, symmetric and transitive relation between two states[30]. It follows that the state space Ω can be partitioned into its *communication classes* $\{\mathcal{C}_1, ..., \mathcal{C}k\}$. The communication classes are mutually disjoint non-empty and their union is Ω. In other words: any two states from the same communication class communicate, states from different communication classes never communicate. We call a communication class $\mathcal{C}_j$ *closed* if the submatrix of P restricted to $\mathcal{C}_j$ has all rows sum to one. Intuitively, if $\mathcal{C}_j$ is closed, then a random walk which enters $\mathcal{C}_j$ will never leave it again. Closed communication classes are also called *recurrent classes*. If a communication class is not recurrent, we call it *transient*. In Theorem 1, we reproduce the statement of Thm. 28 in ref.[30] to compute absorption probabilities towards states that belong to recurrent classes on the Markov chain.

Theorem 1 (Absorption Probabilities) *Consider a MC with transition matrix $P \in \mathbb{R}^{N \times N}$. We may rewrite P as follows:*

$$\begin{bmatrix} \tilde{P} & 0 \\ S & Q \end{bmatrix}, \tag{16}$$

where $\tilde{P}$ and Q are restrictions of P to recurrent and transient states, respectively, and S is the restriction of P to transitions from transient to recurrent states. The upper right 0 is due to the fact that there are no transitions back from recurrent to transient states. Define the matrix $M \in \mathbb{R}^{N \times N}$ via

$$M = (I - Q)^{-1}. \tag{17}$$

Then, the ij-th entry of M describes the expected number of visits of the process to state j before absorption, conditional on the process being initialised in state i. M is often referred to as the fundamental matrix of the MC. Further, the matrix

$$A = (I - Q)^{-1} S \,, \tag{18}$$

in the ij-th entry contains the probability of j being the first recurrent state reached by the MC, given that it was started in i.

For a proof, see See Thm. 26 in ref.[30]. To compute fate probabilities towards the terminal index sets $\mathcal{R}_t$ defined above, we approximate these as recurrent classes, i.e. we remove any outgoing edges from these sets. We then apply Theorem 1, which, for each cell $i \in \mathcal{T}$ yields absorption probabilities towards each of the f cells in each of the n_t recurrent index sets. We aggregate these to yield absorption probabilities towards the recurrent index sets themselves by summing up absorption probabilities towards individual cells in these sets.

CellRank provides an efficient implementation to compute absorption probabilities A naive implementation of absorption probabilities scales cubically in the number of transient cells due to the matrix inversion in Equation (18). The number of transient cells is smaller than the total cell number only by a small constant, so the naive approach can be considered cubic in cell number. This will inevitably fail for large cell numbers. We alleviate this by re-writing Equation (18) as a linear problem,

$$(I - Q)A = S \,. \tag{19}$$

Note that Q is very sparse as it describes transitions between nearest neighbors. Per row, Q has approximately K entries. To exploit the sparsity, iterative solvers are very appealing as their per-iteration cost applied to this problem is linear in cell number and in the number of nearest neighbors. To apply an iterative solver, we must however re-write Equation (19) such that the right hand side is vector valued,

$$(I - Q)a_1 = s_1, \; ..., \; (I - Q)a_{fn_t} = s_{fn_t} \,, \tag{20}$$

where fn_t is the total number of cells which belong to approximately recurrent classes. To solve these individual problems, we use the iterative GMRES[31] algorithm which efficiently exploits sparsity. For optimal performance, we use the PETSc implementation, which makes use of efficient message passing and other practical performance enhancements. Lastly, we parallelize solving the fn_t linear problems. All of these tricks taken together allow us to compute absorption probabilities quickly even for large cell numbers (Fig. 5d and Supplementary Table 1).

1.5 Propagating velocity uncertainty

So far, we have assumed that individual velocity vectors are deterministic, i.e. they have no measurement error. However, this is not correct because RNA velocity is estimated on the basis of spliced and unspliced gene counts, which are noisy quantities. Hence, the velocity vectors v_i themselves should be treated as random variables which follows a certain distribution. Ultimately, our aim is to propagate the distribution in v_i into our final quantities of interest, i.e. state assignments and fate probabilities. However, this is difficult as these final quantities of interest depend on v_i in non-analytical ways, i.e. we cannot write down a closed-form equation which relates the final quantities to v_i. A possible solution to this is to use a Monte Carlo scheme where we draw velocity vectors, compute final quantities based on the draw and repeat this many times. In the limit of infinitely many draws, this will give us the distribution over final quantities, given the distribution in v_i. However, this has the disadvantage that we need to repeat our computation many times, which will get prohibitively expensive for large datasets. To get around this problem, and to allow CellRank to

scale to large datasets, we construct an analytical approximation to the Monte Carlo based scheme. This analytical approximation will only have to be evaluated once and we can omit the sampling. We show in a practical example that the analytical approximation gives very similar results to the sampling based scheme and improves over a deterministic approach by a large margin.

Modeling the distribution over velocity vectors Before we can propagate uncertainty, we need to describe the distribution over velocity vectors, i.e. we need to model the uncertainty present in the velocity vectors which are estimated by scVelo[9] or velocyto[1]. Ideally, we would like these packages themselves to model uncertainty in the raw spliced and unspliced counts and to propagate this into a distribution over velocity vectors. However, as that is currently not being done, we will make an assumption about their distribution and use the KNN graph to fine-tune expectation and variance by considering neighboring velocity vectors. To ease notation and to illustrate the core ideas, we will drop the subscript i in this section and just focus on one fixed cell and it's velocity vector v. Let's assume that v follows a multivariate normal (MVN) distribution,

$$v \sim \mathcal{N}(\mu, \Sigma_v)\,, \tag{21}$$

with mean vector $\mu \in \mathbb{R}^G$ and covariance matrix $\Sigma_v \in \mathbb{R}^{G\times G}$. The MVN is a reasonable choice here as velocities can be both positive and negative and for most genes, as we expect to see both up- and down-regulation, velocity values will be approximately symmetric around their expected value. Let's further assume the covariance matrix to be diagonal, i.e. gene-wise velocities are independent. This is a reasonable assumption to make as gene-wise velocities in both velocyto[1] and scvelo[9] are computed independently. To compute values for μ and Σ_v, consider velocity vector v and its K nearest neighbors. To estimate μ and the diagonal elements of Σ_v, we compute first and second order moments over the velocity vectors of these neighboring cells.

Propagating uncertainty into state assignments and fate probabilities We seek to approximate the expected value of the final quantities of interest (state assignments and fate probabilities), given the distribution in the velocity vectors. Let q be a final quantity of interest. There are two major steps involved in computing q,

$$v \to T \to q\,, \tag{22}$$

where v stands for our inputs, i.e. the velocity vectors, and T is the transition matrix defining the Markov chain. To get from v to T, we evaluate an analytical function which computes correlations and applies a softmax function. We can approximate this first part of the mapping with a Taylor series, which allows us to analytically propagate the distribution in v into T. For the second part of the mapping, we use the expected transition matrix to compute q. This yields an approximation to the expectation of the final quantity we can then compare with the approximation we obtain from a Monte Carlo scheme, which we treat as our ground truth.

Approximating the expected transition matrix In the first step, we compute the expected value of the transition matrix, given the distribution of the velocity vectors. Given a particular draw v from the distribution in Equation (21) and a set of state-change vectors s_k, we compute a vector of probabilities p, which lives on a K-simplex in $\mathbb{R}^K$. Let's denote the mapping from v to p by h,

$$\begin{aligned} h &: \mathbb{R}^G \to R^K \\ v &\mapsto h(v) = p\,. \end{aligned} \tag{23}$$

We can then formulate our problem as finding the expectation of h when applied to v, i.e.

$$\mathrm{E}[h(v)]_{v\sim\mathcal{N}(\mu,\Sigma_v)}\,. \tag{24}$$

To approximate this, expand the i-th component of h in a Taylor-series around μ,

$$h_i(v) = h_i(\mu) + \nabla_v^\top h_i(v)|_\mu (v-\mu) + \frac{1}{2}(v-\mu)^\top \nabla_v^2 h_i(v)|_\mu (v-\mu) + \mathcal{O}(v^3)\,. \tag{25}$$

Define the Hessian matrix of h_i at $v = \mu$ as

$$H^{(i)} = \nabla_v^2 h_i(v)|_\mu\,. \tag{26}$$

Taking the expectation of h_i and using the Taylor-expansion,

$$\mathrm{E}[h_i(v)] \approx h_i(\mu) + \frac{1}{2}\mathrm{E}[(v-\mu)^\top H^{(i)}(v-\mu)]\,. \tag{27}$$

Note that the first order term cancels as $\mathrm{E}[v-\mu] = 0$. The second order term can be further simplified by explicitly writing out the matrix multiplication,

$$\mathrm{E}[(v-\mu)^\top H^{(i)}(v-\mu)] = \sum_{j,k=1}^{G} H^{(i)}_{j,k}\,\mathrm{E}[(v-\mu)_j (v-\mu)_k]\,, \tag{28}$$

where we took the expectation inside the sum and the matrix elements outside the expectation as it does not involve v. For $j \neq i$, the two terms inside the expectation involving v are independent given our distributional assumptions on v and the expectation can be taken separately. Using again the fact that $\mathrm{E}[v-\mu] = 0$, the sum equals zero for $j \neq i$. It follows

$$\sum_{j,k=1}^{G} H^{(i)}_{j,k}\,\mathrm{E}[(v-\mu)_j (v-\mu)_k] = \sum_j H^{(i)}_{j,j}\mathrm{E}[(v_j-\mu_j)^2] = \sum_j H^{(i)}_{j,j}\,\mathrm{var}[v_j]\,. \tag{29}$$

To summarize, our second order approximation to the transition probabilities given the distribution in v reads

$$\mathrm{E}[h_i(v)] \approx h_i(\mu) + \frac{1}{2}\sum_j H^{(i)}_{j,j}\,\mathrm{var}[v_j]\,. \tag{30}$$

We use automatic differentiation as implemented in JAX[32] to compute the Hessian matrices $H^{(i)}$, which ensures they are highly accurate and can be computes in a scalable manner. Further, because we do not hard-code the derivatives, our approach is highly flexible to future changes in the way we compute transition probabilities. If for example it turns out at a later point that an alternative metric works better than Pearson correlation, this is automatically taken care of in the propagation of uncertainties and no changes need to be made, apart from changing the forwards function which computes the transition probabilities themselves. The above procedure can be repeated for all components i and for all cells to yield the second order approximation to the expected transition matrix T, given the distribution over each velocity vector.

Approximating the expected final quantities To arrive at the final quantities of interest, i.e. state assignments and absorption probabilities, we use the expected transition matrix and proceed as in the deterministic case. We validate that this approximation gives very similar results to a fully stochastic approach based on Monte Carlo sampling (Supplementary Fig. 14a,b).

1.6 The CellRank software package

The CellRank software package implements two main modules:

- `kernels` are classes that provide functionality to compute transition matrices based on (directed) single cell data.

- `estimators` are classes that implement algorithms to do inference based on `kernels`. For example, `estimators` compute macrostates and fate probabilities.

This modular and object oriented design allows CellRank to be extended easily into two directions. On the one hand, including more kernels to take into account further means of directional single cell data such as metabolic labeling or experimental time. On the other hand, including more estimators to learn new abstractions of cellular dynamics. The kernel module currently implements a

- `VelocityKernel` which computes a transition matrix on the basis of a KNN graph and RNA velocity information.
- `PalantirKernel` which mimicks the original routine outlined in the Palantir[8] paper to compute a directed transition matrix on the basis of a KNN graph and a pseudotime.
- `ConnectivityKernel` which takes the adjacency matrix underlying the KNN graph and row-normalizes it to obtain a valid transition matrix. This is essentially the transition matrix used in the DPT[10] algorithm.
- `PrecomputedKernel` which accepts any pre-computed transition matrix and allows for easy interfacing with the CellRank software.

All `kernel` classes are derived from a `base kernel` class which implements density normalization as implemented in ref.[10]. Instances of `kernel` classes can be combined by simply adding them up using the + operator, potentially including weights. A typical code snippet to compute a transition matrix will look like this:

```
from cellrank.tools.kernels import VelocityKernel, ConnectivityKernel

vk = VelocityKernel(adata).compute_transition_matrix()
ck = ConnectivityKernel(adata).compute_transition_matrix()

combined_kernel = 0.9*vk + 0.1*ck
```

The estimator module currently implements a

- `CFLARE` estimator. `CFLARE` stands for *Clustering and Filtering of Left and Right Eigenvectors*. This estimator computes terminal states directly by filtering cells in the top left eigenvectors and clustering them in the top right eigenvectors, thereby combining ideas of spectral clustering and stationary distributions.
- `GPCCA` estimator. The GPCCA estimator.

All `estimator` classes are derived from a `base estimator` class which allows to compute fate probabilities, regardless of how terminal/intermediate states have been computed. A typical code snippet to compute macrostates and fate probabilities will look like this:

```
from cellrank.tools.estimators import GPCCA

# initialise the estimator
gpcca = GPCCA(combined_kernel)

# compute macrostates and identify the terminal states among them
gpcca.compute_macrostates()
gpcca.compute_terminal_states()

# comptue fate probabilities
gpcca.compute_absorption_probabilities()
```

Both `kernels` and `estimators` implement a number of plotting functions to conveniently inspect results. We designed CellRank to be highly scalable to ever increasing cell numbers, widely applicable and extendable to problems in single cell dynamical inference, user friendly with tutorials and comprehensive documentation and robust with 88 % code coverage. CellRank is open source, fully integrated with SCANPY and scVelo and freely available at `https://cellrank.org`.

2 Computing a directed PAGA graph

Partition-based graph abstraction (PAGA)[33] provides an interpretable graph-like connectivity map of the data manifold. It is obtained by associating a node with each manifold partition (e.g. cell type) and connecting each node by weighted edges that represent a statistical measure of connectivity between the partitions. The model considers groups/nodes as connected if their number of inter-edges exceeds what would have been expected under random assignment. The connection strength can be interpreted as confidence in the presence of an actual connection and allows discarding spurious, noise-related connections.

Here, we extend PAGA by directing the edges as to reflect the RNA velocity vector field rather than transcriptome similarity. The connectivity strengths are defined based on the velocity graph. That is, for each cell correlations between the cell's velocity vector and its potential, cell-to-cell transitions are computed (Subsection 1.2). Inter-edges are considered whose correlation passes a certain threshold (default: 0.1). The number of inter-edges are then tested against random assignment for significance.

To further constrain the single cell graph, compute a gene-shared latent time using scVelo[34]. In short, this aggregates the per-gene time assignments computed in scVelo's dynamical model to a global scale which faithfully approximates a single-cells internal clock. Once we have computed the initial states using CellRank, we can use these as a prior for latent time to force it to start in this state. All of latent time, initial and terminal states can in turn be used as a prior to regularize the directed graph. At single-cell level, we use latent time as a constraint to prune the cell-to cell transition edges to those that match the ordering of cells given by latent time. For the initial and terminal states, the edges are further constrained to only retain those cell-to-cell transitions that constitute outgoing flows for cells in initial cellular populations, and to incoming flows for cells in terminal populations.

Finally, a minimum spanning is constructed for the directed abstracted graph. It is obtained by pruning node-to-node edges such that only the most confident path from one node to another is retained. If there are multiple paths to reach a particular node, only the path with the highest confidence is kept.

3 Computing gene expression trends along lineages

CellRank computes fate probabilities which specify how likely each individual cell is to transition towards each identified terminal state (Subsection 1.4). Combined with any pseudo-temporal measure like DPT[10], scVelo's latent time[2] or Palantir's pseudotime[35], this allows us to compute and to compare gene expression trends towards specific terminal populations. In contrast to other methods, we do not partition the set of all cells into clusters and define lineage each lineage via an ordered set of clusters. Instead, we use all cells to fit each lineage but we weigh each cell according to its fate probability, our measure of lineage membership. This means that for cells uncommitted between two or more fates, we allow them to contribute to each one of these, weighted by the fate probabilities. For cells committed towards any particular fate, their fate probabilities towards the remaining fates will be zero or almost zero which naturally excludes them when fitting these other lineages.

Imputing gene expression recovers trends from noisy data To improve the robustness and resolution of gene expression trends, we adapt two strategies. First, we use imputed gene expression values and second, we fit Generalized Additive Models (GAMs). For gene expression imputation, we use MAGIC[36] by default, however, any imputed gene expression matrix can be supplied. MAGIC is based on KNN imputation and makes use of the covariance structure among neighboring cells to estimate expression levels for each gene. The KNN graph is computed globally, based on the expression values of all genes and not just the one we are currently considering.

Generalized Additive Models (GAMs) robustly fit gene expression values While sliding window approaches are known to be sensitive to local density differences and only take into account the current gene when determining gene expression trends, we fit GAMs to gene expression values which have been imputed borrowing information from neighboring cells via a KNN graph. Using GAMs allows us to flexibly model many different kinds of gene trends in a robust and scalable manner. We fit the gene expression trend for branch t in gene g via

$$y_{gi} = \beta_0 + f(\tau_i)\ \forall i : F_{ib} > 0\,, \tag{31}$$

where y_{gi} is gene expression of gene g in cell i, τ_i is the pseudotemporal value of cell i and F is the fate matrix of Subsection 1.4. By default, we use cubic splices for the smoothing functions f as these have been shown to be effective in capturing non-linear relationships in trends[37].

To visualize the smooth trend, we select 200 equally spaced testing points along pseudotime and we predict gene expression at each of them using the fitted model of Equation (31). To estimate uncertainty along the trend, we use the standard deviation of the residuals of the fit, given by

$$\sigma_{\hat{y}_p} = \sqrt{\frac{\sum_{j=1}^{n}(y_j - \hat{y_j})^2}{n-2}}\sqrt{1 + \frac{1}{n} + \frac{(\tau_p - \bar{\tau})^2}{\sum_{j=1}^{n}(\tau_j - \bar{\tau})^2}}\,, \tag{32}$$

where $\hat{y}_p$ denotes predicted gene expression at test point p, $\bar{\tau}$ denotes average pseudo-time across all cells and n is the number of test points[38]. For the fitting of Equation (31), we provide interfaces to both the R package `mgcv`[39] as well as the Python package `pyGAM`[40]. We parallelize gene fitting to scale well in the number of genes, which is important when plotting heatmaps summarizing many gene expression trends.

Visualizing gene expression trends for the pancreas example For CellRank's gene expression trends of lineage-associated genes along the alpha, beta, epsilon and delta fates, we used Palantir's pseudotime[35], MAGIC imputed data[36] and the `pyGAM`[40] package to fit GAMs. We used default values to fit the splines, i.e. we place 10 knots along pseudotime and we use cubic splines. For the delta lineage, fate probabilities among early cells were very low (0.01 average fate probability among Ngn3 high EP cells, see Fig. 2e). This reflects the small size of the delta population (70 cells or 3% of the data, see Supplementary Fig. 10a,b) as well as the fact that delta cells are produced mostly at later stages in pancreatic development[41]. To still be able to reliably fit gene expression of early cells along the delta lineage, we thresholded weights at 0.05, i.e. weights smaller than this value were clipped to this value. This was done only for the fitting of gene expression trends.

4 Clustering gene expression trends

CellRank allows gene expression trends along a particular lineage to be clustered, thus recovering the major patterns of regulation towards a specific terminal state like (transient) up- or down-regulation. For the set of genes we are interested in, we recover their regulation along a specific lineage by fitting GAMs in pseudotime where we supply fate probabiliteis as cell-level lineage weights (Section 3). In the next step, we cluster the GAM-smoothed gene expression trends. For this, we z-transform

expression values and we compute a PCA representation of the trends. By default, we use 50 PCs. We then compute a KNN graph in PC space as outlined in Subsection 1.2 and we cluster the KNN graph using the louvain[42] or leiden[43] algorithms. For each recovered cluster, we compute its mean and standard deviation (point-wise, for all testing points that were used for smoothing) and visualize them, together with the individual, smoothed trends per cluster. As gene-trend fitting is efficiently parallelized in CellRank, such an analysis can be performed in an unbiased fashion for large gene sets. For 10k genes, the run-time is about 6 min on a 2019 Mac book pro with 2,8 GHz Intel Core i7 processor and 16 GB RAM.

Clustering gene expression trends towards the delta fate To cluster gene expression trends towards the delta fate in Fig. 3e, all genes which were expressed in at least 10 cells were included (12,987 genes). Smooth gene expression trends along the delta lineage were determined using Palantir's pseudotime[35]. We used $K = 30$ nearest neighbors for the gene-trend KNN graph and the louvain algorithm with resolution parameter set to 0.2 to avoid over-clustering the trends.

5 Uncovering putative driver genes

To find genes which are expressed at high levels in cells that are biased towards a particular fate, we compute Person's correlation between expression levels of a set of genes and fate probabilities. We sort genes according to their correlation values and consider high-scoring genes as candidate driver genes. By default, we consider all genes which have passed pre-processing gene filtering thresholds. The computation of correlation values can be restricted to a set of pre-defined clusters if one is interested in driver genes which act in a certain region of the phenotypic manifold.

Uncovering putative driver genes for delta development To uncover putative driver genes towards the delta fate in Fig. 3d,e, we considered 12,987 genes which were expressed in at least 10 cells. We computed correlation of total-count normalized, log transformed gene expression values with the probability of becoming a delta cell. We restricted correlation computation to the Fev+ cluster, where we expected the fate decision towards delta to occur.

6 Robustness analysis

We were interested in evaluating how much CellRank's fate probabilities change in response to changes in the following key pre-processing parameters:

- the number of neighbors K used for KNN graph construction (Subsection 1.2)
- scVelo's gene-filtering parameter `min_shared_counts` which determines how many counts a gene must have in both spliced and unspliced layers
- scVelo's gene filtering parameter `n_top_genes` which determines the number of most highly variable genes used for the velocity computation
- the number of principal components `n_pcs` used for KNN graph construction (Subsection 1.2)

In addition to the 4 key pre-processing parameters, we were interested to see how much CellRank's results change when we randomly sub-sample the number of cells to 90% of the original cell number and when we vary the number of macrostates. We used the pancreas example[16] in all of the following comparisons.

Robustness with respect to key pre-processing parameters To evaluate robustness with respect to changes in the pre-processing parameters, we varied one parameter at a time (keeping the others

fixed), computed macrostates and fate probabilities towards them. We then compared the fate probabilities for different values of the parameter by computing pairwise Pearson correlation among all possible pairs of values we used for the parameter. We did this separately for each lineage, i.e. for the alpha, beta, epsilon and delta lineages. For each lineage, we recorded the median and minimum correlation achieved across all the different comparisons. We always computed enough macrostates so that the alpha, beta, epsilon and delta states were included. Naturally, the precise location of the terminal states changed slightly across parameter combinations. For this reason, the correlation values we recorded reflect robustness of the entire CellRank workflow, including both the computation of terminal states as well as fate probabilities. In a separate comparison, we were interested in evaluating the robustness of just the last step of the CellRank algorithm, i.e. the computation of fate probabilities. For this, we kept the terminal states fixed across parameter variations and proceeded as above otherwise, computing pairwise Pearson correlations among fate probabilities per lineage across all parameter value combinations. Furthermore, we were interested to see whether CellRank's robustness changes when we propagate uncertainty. For this, we repeated all the aforementioned computations using our analytical approximation to propagate uncertainty.

Robustness with respect to random sub-sampling of cells We subsampled the data to 90% of cells, computed macrostates and fate probabilities towards the alpha, beta, epsilon and delta states. We repeated this 20 times, recorded all computed fate probabilities and compared them pairwise per lineage using Pearson's correlation for all possible pairs of random draws. As in the above evaluation for the key pre-processing parameters, we recorded minimum and median correlation per lineage across all pairs and we repeated this for fixed terminal states and for propagated uncertainty.

Robustness with respect to the number of macrostates To evaluate sensitivity with respect to this parameter, we varied the number of macrostates between 10 and 16 and confirmed that within this range, the key terminal and initial states exist and remain in the same location.

7 Pancreas data example

We used a scRNA-seq time-series data set comprising embryonic days $12.5 - 15.5$ of pancreatic development in mice assayed using 10x Genomics[16]. We restricted the data to the last time point (E15.5) and to the Ngn3 low EP, Ngn3 high EP, Fev+ and endocrine clusters to focus on the late stages of endocrinogenesis where all of alpha, beta, epsilon and delta fates are present. We further filtered out cycling cells to amplify the differentiation signal. Our final subset contained 2531 cells. We kept the original cluster annotations which were available on a coarse level and on a fine level. On the fine level, the Fev+ cluster was sub-clustered into different populations which are biased towards different endocrine fates (Supplementary Fig. 10c).

Data pre-processing and velocity computation For the following processing, we used scVelo[9] and SCANPY[13] with mostly default parameters. Loom files containing raw spliced and unspliced counts were obtained by running the velocyto[1] command-line pipeline. We filtered genes to be expressed in at least 10 cells and to have at least 20 counts in both spliced and unspliced layers. We further normalized by total counts per cell, log transformed the data and kept the top 2000 highly variable genes. We then computed a PCA representation of the data and used the top 30 PCs to compute a KNN graph with $K = 30$ nearest neighbors. For velocity computation, we used scVelo's dynamical model of splicing kinetics. We evaluate robustness of CellRank's results to changes in these pre-processing parameters (Section 6).

Embedding computation We used the KNN graph to compute a PAGA[7] representation of the data. The PAGA graph was used to initialize the computation of a UMAP[11,44] representation of

the data. Note that UMAP was only used to visualize the data and was not supplied to CellRank to compute the transition matrix or any downstream quantities.

CellRank parameters We use CellRank's analytical stochastic approximation to compute transition probabilities and include a diffusion kernel with weight 0.2 (Subsection 1.2 and Subsection 1.5). We compute 12 macrostates and automatically detect the terminal alpha, beta and epsilon states. The delta population is picked up automatically as a macrostate. We manually assign it the terminal label.

8 Lung data example

We used a scRNA-seq time-series data-set of lung regeneration past bleomycin injury in mice assayed using Dropseq[15,45]. The data-set contained 18 time points comprising days 0-54 past injury. There was daily sampling from days 2-13 and wider time-lags between the following time-points. Two replicate mice were used per time point. We restricted the data to days 2-15 to make sure that the sampling is dense enough for velocities to be able to meaningfully extrapolate gene expression. If time points are too far apart, then RNA velocity cannot be used to predict the next likely cellular state because the linear extrapolation is only meaningful on the time scales of the splicing kinetics. Our final subset contained 24,882 cells. We kept the original cluster annotations.

Data pre-processing and velocity computation For the following processing, we used scVelo[9] and SCANPY[13] with mostly default parameters. Loom files containing raw spliced and unspliced counts were obtained by running the velocyto[1] command-line pipeline. We filtered genes to be expressed in at least 10 cells and to have at least 20 counts in both spliced and unspliced layers. We further normalized by total counts per cell, log transformed the data and kept the top 2000 highly variable genes. We kept the PCA coordinates from the original study and computed a KNN graph with $K = 30$ nearest neighbors using the top 50 PCs. For velocity computation, we used scVelo's dynamical model of splicing kinetics.

Embedding computation The lung data was processed in three separate batches. We used BBKNN[46] to compute a batch corrected KNN graph with 10 neighbors within each batch. The corrected KNN graph was used to compute a UMAP[11,44] representation of the data. Note that UMAP was only used to visualize the data and was not supplied to CellRank to compute the transition matrix or any downstream quantities. We did not use BBKNN to correct the graph we used for velocity computation as it is an open question how to do batch correction for velocity computation. We used uncorrected data for velocity computation.

CellRank parameters We use CellRank's analytical stochastic approximation to compute transition probabilities and include a diffusion kernel with weight 0.2 (Subsection 1.2 and Subsection 1.5). On the full data of Fig. 6a, we compute 9 macrostates. On the reduced data of Fig. 6d, we compute 3 macrostates.

Defining stages of the differentiation trajectory We sub-setted cells to goblet and basal cells and re-run CellRank on the subset to investigate the trajectory at higher resolution. CellRank automatically detected initial and terminal states and computed fate probabilities towards the terminal states (Supplementary Fig. 24a-c). Further, we applied Palantir[35] to the subset to compute a pseudotime (Supplementary Fig. 24d,e). We combined the pseudotime with CellRank's fate probabilities to define three stages of the dedifferentiation trajectory by requiring cells to have at least 0.66 basal probability. Cells passing this threshold were assigned to three bins of equal size along the pseudotemporal axis. We used this binning to define the three stages of the trajectory.

9 Methods comparison

We compared CellRank with the following similarity-based tools that compute probabilistic fate assignments on the single cell level: Palantir[35], FateID[47] and STEMNET[48]. We compared these methods in terms of the identification of initial and terminal states, fate probabilities, gene expression trends and run-time.

The Palantir algorithm Palantir[35] computes a KNN graph in the space of diffusion components and uses this graph to compute a pseudotime via iteratively updating shortest path distances from a set of waypoints. Palantir required us to provide a number of *waypoint cells* - essentially a smaller number of cells that the system is reduced to in order to make it computationally feasible. We set this number to 1200 cells for the pancreas data and to 15% of the total cell number of the runtime and memory benchmarks on the reprogramming dataset[49] below. For pseudotime computation, an initial cells needs to be supplied by the user. The pseudotime is used to direct edges in the KNN graph by removing edges that point from later cells to earlier cells in pseudotime. The stationary distribution of the resulting directed transition matrix is combined with extrema in the diffusion components to identify terminal cells. Absorption probabilities towards the terminal cells serve as fate probabilities. Gene expression trends are computed similarly to CellRank, by fitting GAMs in pseudotime where each cell contributes to each lineage according to its fate probabilities.

The FateID algorithm FateID[47] either requires the user to provide terminal populations directly or through a set of marker genes. Terminal populations are used to train a random forest classifier. The classifier is applied to a set of cells in the neighborhood of each terminal cluster where it predicts the likely fate of these cells. The training set is iteratively expanded and the Random forest is re-trained on expanding populations, thus moving from the committed populations backward in time, classifying the fate of increasingly earlier cells. Two key parameters here are the size of the training and test sets used for the Random forest classifier, which we set to 1% of the data in all benchmarks. Gene expression trends are computed by selecting a (discrete) set of cells which pass a certain threshold for fate bias towards a specific terminal population. A principal curve is fit to these cells in a low dimensional embedding and pseudotime is assigned via projection onto this curve. Alternatively, the authors recommend to compute diffusion pseudotime[10] (DPT) on the set of cells selected for a particular lineage. Gene expression values are then normalised and a local regression (LOESS) is performed to obtain mean trends. In contrast to CellRank and Palantir, this approach does not provide confidence intervals for the expression trends, it is dependent on low dimensional embeddings (principal curve fit) and it discreetly assigns cells to lineages, thereby ignoring the gradual nature of fate commitment when visualizing gene expression trends. Since different cells are selected for different lineages, the computed pseudo-temporal orderings are incompatible and gene trends along different lineages cannot be visualized jointly.

The STEMNET algorithm STEMNET[48] requires the user to provide the terminal populations directly as input to the algorithm. It then trains an elastic-net regularized generalized linear model on the terminal populations to predict state membership. This first step serves as feature selection - it selects a set of genes which are specific to their terminal populations. In the next step, the classifier uses expression of these genes to predict fate bias for the remaining, transient cells. STEMNET uses the computed fate probabilities to place cells on a simplex in 2 dimensions as a dimensionality reduction method. It does not offer a method to visualize gene expression trends.

Fate probabilities In order to enable a fair comparison across methods, we supplied all methods with CellRank's identified terminal states and compared predicted fate probabilities. Methods differed in the format they require terminal state information to be passed: for Palantir, we passed individual cells, i.e. 4 cells, one for each of the three terminal states and one initial cell from the initial

state. For STEMNET, we passed populations of cells defined through the underlying transcriptomic clusters. We passed the alpha, beta, epsilon and delta clusters defined through the sub-clustering of Supplementary Fig. 10c. For FateID, we passed marker genes to identify the terminal populations. For each terminal state, we passed its corresponding hormone-production associated gene, i.e. *Ins1* for beta, *Gcg* for alpha, *Ghrl* for epsilon and *Sst* for delta[50]. We checked whether methods correctly predicted beta to be the dominant fate among early cells by computing the average fate prediction among Ngn3 high EP cells.

Gene expression trends To visualize gene expression trends, we used the functionalities that each method provided. STEMNET did not have an option to compute gene expression trends. For CellRank, we visualized gene expression trends as described in Section 3. For Palantir, we used default parameters. For FateID, it was difficult to find a good threshold value to assign cells to lineages. If this value is too high, then early cells in the trajectory are not selected and the terminal states are isolated. If this value is too low, then for a subset of the lineages, very unlikely cells are assigned and the trends are very unspecific. The default value is 0.25, which was too high in our case. We decided to set the threshold at 0.15, which was a compromise between trying to have early cells in every lineage and making sure that irrelevant cells are not assigned. We computed DPT[10] on the set of the selected cells, as recommended in the original publication. To identify a root cell for each lineage, we first computed DPT on the entire data-set, then subsetted to a lineage-specific set of cells and picked the cell with the earliest original DPT value as the root cell for the second DPT computation. We visualized expression trends for the key lineage drivers *Pax4*[51] and *Pdx1*[52–54] (beta), *Arx*[51] (alpha), *Ghrl*[50] (epsilon) and *Hhex*[55] and *Cd24a*[56,57] (delta) as well as the lineage accociated genes *Peg10*[58,59] (alpha) and *Irs4*[59] (epsilon). In Fig. 5c, we checked whether methods correctly predicted upregulation of *Pdx1* along the beta fate.

Runtime We compared run-time of the four methods applied to a scRNA-seq dataset comprising 100k cells undergoing reprogramming from mouse embryonic fibroblasts to induced endoderm progenitor cells[49]. We randomly subsampled the data-set to obtain 10 data-sets of increasing size, starting from 10k cells in steps of 10k until 100k cells. For each sub-sampled dataset, we applied each method 10 times and computed the mean runtime as well as the standard error on the mean.

We used CellRank to compute 3 terminal states and we supplied these to all other methods to ensure that the number of terminal states is consistent across methods. Methods differed in the format they require terminal state information to be passed: for Palantir, we passed individual cells, i.e. three terminal cells and one initial cell (taken from the earliest time point of the reprogramming data). For STEMNET, we passed a set of cells for each terminal state by choosing the cells which have been most confidently assigned to each terminal state by CellRank. For each terminal state, we passed a number of cells that was equal to 1% of the total cell number. FateID requires marker genes to identify the terminal populations, so we computed the top 3 lineage drivers per CellRank-identified terminal state and passed these.

For CellRank, we separately recorded the time it took to compute the terminal states and fate probabilities. For terminal states in CellRank, we included in this benchmark the entire workflow from computing the transition matrix via decomposing it into macrostates to identifying the terminal states among the macrostates. For fate probabilities, we benchmarked the `compute_absorption_probabilities()` method (CellRank), the `run_palantir()` function (Palantir), the `fateBias()` function (FateID) and the `runSTEMNET()` function (FateID).

Comparisons were run on an Intel(R) Xeon(R) Gold 6126 CPU @ 2.60GHz and 32 cores. Each job was allocated at least 90 GiB RAM and we recorded the actual peak memory usage (see below). FateID did not finish on 100k cells because of a memory error due to densification of a large matrix.

Peak memory usage The setup was identical to the setup for the runtime comparison above, only that we recorded peak memory usage of each method (Supplementary Table 2). For the Python-based methods CellRank and Palantir, we used the `memory-profiler`[60] package whereas for the R-based packages STEMNET and FateID, we used the peakRAM[61] profiler. CellRank and Palantir efficiently parallelize their computations across several cores which increases their peak memory consumption. We repeated our evaluation for these two methods on 100k cells using just a single core to estimate the size of this effect (Supplementary Table 3).

10 Immunofluorescence stainings and microscopy on airway epithelial cells

Formalin-fixed paraffin-embedded lung sections (3.5 μm thick) from bleomycin-treated mice at day 10 (n=2) and day 22 (n=2) after bleomycin instillation, and from phosphate-buffered saline (PBS)-treated controls (n=2) were stained as previously described[15]. In brief, after deparaffinization, rehydration and heat-mediated antigen retrieval with citrate buffer (10 mM, pH = 6.0), sections were blocked with 5% bovine serum albumin for 1 h at room temperature and then incubated with the following primary antibodies overnight at 4°C: rabbit anti-Bpifb1 (kindly provided by C. Bingle[62], 1:500), mouse anti-Trp63 (abcam, ab735, clone A4A, 1:50) and chicken anti-Krt5 (BioLegend, Poly9059, 1:1,000).

For visualization of stainings the following secondary antibodies were used: Goat anti-rabbit Alexa Fluor® 488 (Invitrogen, A11008, 1:250), Goat anti-chicken Alexa Fluor® 568 (Invitrogen, A11041,1:250) and goat anti-mouse Alexa Fluor® 647 (Invitrogen, A21236, 1:250). Cell nuclei were visualized with 4',6-diamidino-2-phenylindole (DAPI).

Immunofluorescent images were acquired with an AxioImager.M2 microscope (Zeiss) using a Plan-Apochromat 20x/0.8 M27 objective. For quantification of immunofluorescence stainings, five different intrapulmonary regions were recorded per mouse and the percentage of positively stained cells normalized to the total number of airway cells was manually quantified using Fiji software (ImageJ, v. 2.0.0).

References

[1] Gioele La Manno, et al. RNA velocity of single cells. *Nature*, page 1, August 2018. ISSN 1476-4687. doi: 10.1038/s41586-018-0414-6. URL https://www.nature.com/articles/s41586-018-0414-6.

[2] Volker Bergen, et al. Generalizing RNA velocity to transient cell states through dynamical modeling. *Nature Biotechnology*, pages 1–7, August 2020. ISSN 1546-1696. doi: 10.1038/s41587-020-0591-3. URL https://www.nature.com/articles/s41587-020-0591-3. Publisher: Nature Publishing Group.

[3] Florian Erhard, et al. scSLAM-seq reveals core features of transcription dynamics in single cells. *Nature*, 571(7765):419, July 2019. ISSN 1476-4687. doi: 10.1038/s41586-019-1369-y. URL https://www.nature.com/articles/s41586-019-1369-y.

[4] Gert-Jan Hendriks, et al. NASC-seq monitors RNA synthesis in single cells. *Nature Communications*, 10(1):3138, July 2019. ISSN 2041-1723. doi: 10.1038/s41467-019-11028-9. URL https://www.nature.com/articles/s41467-019-11028-9. Number: 1 Publisher: Nature Publishing Group.

[5] Nico Battich, et al. Sequencing metabolically labeled transcripts in single cells reveals mRNA turnover strategies. *Science*, 367(6482):1151–1156, March 2020. ISSN 0036-8075, 1095-9203. doi: 10.1126/science.aax3072. URL https://science.sciencemag.org/content/367/6482/1151. Publisher: American Association for the Advancement of Science Section: Report.

[6] Qi Qiu, et al. Massively parallel, time-resolved single-cell RNA sequencing with scNT-Seq. *bioRxiv*, page 2019.12.19.882050, January 2020. doi: 10.1101/2019.12.19.882050. URL https://www.biorxiv.org/content/10.1101/2019.12.19.882050v2. Publisher: Cold Spring Harbor Laboratory Section: New Results.

[7] F. Alexander Wolf, et al. PAGA: graph abstraction reconciles clustering with trajectory inference through a topology preserving map of single cells. *Genome Biology*, 20(1):59, March 2019. ISSN 1474-760X. doi: 10.1186/s13059-019-1663-x. URL https://doi.org/10.1186/s13059-019-1663-x.

[8] Manu Setty, et al. Palantir characterizes cell fate continuities in human hematopoiesis. *bioRxiv*, page 385328, August 2018. doi: 10.1101/385328. URL https://www.biorxiv.org/content/early/2018/08/05/385328.

[9] Volker Bergen, et al. Generalizing RNA velocity to transient cell states through dynamical modeling. *bioRxiv*, page 820936, October 2019. doi: 10.1101/820936. URL https://www.biorxiv.org/content/10.1101/820936v1.

[10] Laleh Haghverdi, et al. Diffusion pseudotime robustly reconstructs lineage branching. *Nature Methods*, 13(10):845–848, October 2016. ISSN 1548-7105. doi: 10.1038/nmeth.3971. URL https://www.nature.com/articles/nmeth.3971.

[11] Leland McInnes and John Healy. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. *arXiv:1802.03426 [cs, stat]*, February 2018. URL http://arxiv.org/abs/1802.03426. arXiv: 1802.03426.

[12] David I Spivak. METRIC REALIZATION OF FUZZY SIMPLICIAL SETS. page 4, 2009.

[13] F. Alexander Wolf, et al. SCANPY: large-scale single-cell gene expression data analysis. *Genome Biology*, 19(1):15, 2018. ISSN 1474-760X. doi: 10.1186/s13059-017-1382-0.

[14] R. R. Coifman, et al. Geometric diffusions as a tool for harmonic analysis and structure definition of data: Diffusion maps. *Proceedings of the National Academy of Sciences of the United States of America*, 102(21):7426–7431, May 2005. ISSN 0027-8424, 1091-6490. doi: 10.1073/pnas.0500334102. URL http://www.pnas.org/content/102/21/7426.

[15] Maximilian Strunz, et al. Alveolar regeneration through a Krt8+ transitional stem cell state that persists in human lung fibrosis. *Nature Communications*, 11(1):3559, July 2020. ISSN 2041-1723. doi: 10.1038/s41467-020-17358-3. URL https://www.nature.com/articles/s41467-020-17358-3. Number: 1 Publisher: Nature Publishing Group.

[16] Aimée Bastidas-Ponce, et al. Comprehensive single cell mRNA profiling reveals a detailed roadmap for pancreatic endocrinogenesis. *Development*, 146(12):dev173849, June 2019. ISSN 0950-1991, 1477-9129. doi: 10.1242/dev.173849. URL http://dev.biologists.org/lookup/doi/10.1242/dev.173849.

[17] Charlotte Soneson, et al. Preprocessing choices affect RNA velocity results for droplet scRNA-seq data. *bioRxiv*, page 2020.03.13.990069, March 2020. doi: 10.1101/2020.03.13.990069. URL https://www.biorxiv.org/content/10.1101/2020.03.13.990069v1. Publisher: Cold Spring Harbor Laboratory Section: New Results.

[18] Bushra Raj and Benjamin J. Blencowe. Alternative Splicing in the Mammalian Nervous System: Recent Insights into Mechanisms and Functional Roles. *Neuron*, 87(1):14–27, July 2015. ISSN 0896-6273. doi: 10.1016/j.neuron.2015.05.004. URL https://www.cell.com/neuron/abstract/S0896-6273(15)00411-0. Publisher: Elsevier.

[19] Nicole M. Martinez and Kristen W. Lynch. Control of alternative splicing in immune responses: many regulators, many predictions, much still to learn. *Immunological Reviews*, 253(1):216–236, 2013. ISSN 1600-065X. doi: 10.1111/imr.12047. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/imr.12047. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/imr.12047.

[20] Qun Pan, et al. Deep surveying of alternative splicing complexity in the human transcriptome by high-throughput sequencing. *Nature Genetics*, 40(12):1413–1415, December 2008. ISSN 1546-1718. doi: 10.1038/ng.259. URL https://www.nature.com/articles/ng.259. Number: 12 Publisher: Nature Publishing Group.

[21] Eric T. Wang, et al. Alternative isoform regulation in human tissue transcriptomes. *Nature*, 456(7221):470–476, November 2008. ISSN 1476-4687. doi: 10.1038/nature07509. URL https://www.nature.com/articles/nature07509. Number: 7221 Publisher: Nature Publishing Group.

[22] Bernhard Reuter, et al. Generalized Markov State Modeling Method for Nonequilibrium Biomolecular Dynamics: Exemplified on Amyloid Conformational Dynamics Driven by an Oscillating Electric Field. *Journal of Chemical Theory and Computation*, 14(7):3579–3594, July 2018. ISSN 1549-9618, 1549-9626. doi: 10.1021/acs.jctc.8b00079. URL https://pubs.acs.org/doi/10.1021/acs.jctc.8b00079.

[23] Bernhard Reuter, et al. Generalized Markov modeling of nonreversible molecular kinetics. *The Journal of Chemical Physics*, 150(17):174103, May 2019. ISSN 0021-9606. doi: 10.1063/1.5064530. URL https://aip.scitation.org/doi/10.1063/1.5064530. Publisher: American Institute of Physics.

[24] Susanna Röblitz and Marcus Weber. Fuzzy spectral clustering by PCCA+: application to Markov state models and data classification. *Advances in Data Analysis and Classification*, 7(2):147–179, June 2013. ISSN 1862-5347, 1862-5355. doi: 10.1007/s11634-013-0134-6. URL http://link.springer.com/10.1007/s11634-013-0134-6.

[25] Gene H. Golub and Charles F. Van Loan. *Matrix computations*. Johns Hopkins studies in the mathematical sciences. The Johns Hopkins University Press, Baltimore, fourth edition edition, 2013. ISBN 978-1-4214-0794-4. OCLC: ocn824733531.

[26] Roger A. Horn and Charles R. Johnson. *Matrix analysis*. Cambridge University Press, Cambridge ; New York, 2nd ed edition, 2012. ISBN 978-0-521-83940-2.

[27] Martin K. Scherer. msmtools: MSMTools. URL http://github.com/markovmodel/msmtools.

[28] Vicente Hernandez, et al. SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Transactions on Mathematical Software*, 31(3):351–362, September 2005. ISSN 0098-3500. doi: 10.1145/1089014.1089019. URL https://doi.org/10.1145/1089014.1089019.

[29] Lisandro D. Dalcin, et al. Parallel distributed computing using Python. *Advances in Water Resources*, 34(9):1124–1139, September 2011. ISSN 0309-1708. doi: 10.1016/j.advwatres.2011.04.013. URL http://www.sciencedirect.com/science/article/pii/S0309170811000777.

[30] Anders Tolver. An introduction to Markov chains. 2016.

[31] Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on scientific and statistical computing*, 7(3):856–869, 1986.

[32] Roy Frostig, et al. Compiling machine learning programs via high-level tracing. page 3.

[33] F. Alexander Wolf, et al. Graph abstraction reconciles clustering with trajectory inference through a topology preserving map of single cells. *bioRxiv*, page 208819, October 2017. doi: 10.1101/208819. URL https://www.biorxiv.org/content/early/2017/10/25/208819.

[34] Koen Van den Berge, et al. Trajectory-based differential expression analysis for single-cell sequencing data. *bioRxiv*, page 623397, May 2019. doi: 10.1101/623397. URL https://www.biorxiv.org/content/10.1101/623397v1.

[35] Manu Setty, et al. Characterization of cell fate probabilities in single-cell data with Palantir. *Nature Biotechnology*, 37(4):451, April 2019. ISSN 1546-1696. doi: 10.1038/s41587-019-0068-4. URL https://www.nature.com/articles/s41587-019-0068-4.

[36] David van Dijk, et al. Recovering Gene Interactions from Single-Cell Data Using Data Diffusion. *Cell*, 174 (3):716–729.e27, July 2018. ISSN 00928674. doi: 10.1016/j.cell.2018.05.061. URL https://linkinghub.elsevier.com/retrieve/pii/S0092867418307244.

[37] Trevor Hastie and Robert Tibshirani. Generalized Additive Models. *Statistical Science*, 1(3):297–310, August 1986. ISSN 0883-4237, 2168-8745. doi: 10.1214/ss/1177013604. URL https://projecteuclid.org/euclid.ss/1177013604. Publisher: Institute of Mathematical Statistics.

[38] Joseph S. DeSalvo. Standard Error of Forecast in Multiple Regression: Proof of a Useful Result. January 1970. URL https://www.rand.org/pubs/papers/P4365.html. Publisher: RAND Corporation.

[39] Simon Wood. mgcv: Mixed GAM Computation Vehicle with Automatic Smoothness Estimation, August 2020. URL https://CRAN.R-project.org/package=mgcv.

[40] Daniel Servén and Charlie Brummit. pyGAM: Generalized Additive Models in Python, October 2018. URL https://zenodo.org/record/1476122#.X2nIBpMza3I.

[41] Kerstin A. Johansson, et al. Temporal Control of Neurogenin3 Activity in Pancreas Progenitors Reveals Competence Windows for the Generation of Different Endocrine Cell Types. *Developmental Cell*, 12(3):457–465, March 2007. ISSN 1534-5807. doi: 10.1016/j.devcel.2007.02.010. URL http://www.sciencedirect.com/science/article/pii/S1534580707000615.

[42] Vincent D. Blondel, et al. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

[43] Vincent Traag, et al. From Louvain to Leiden: guaranteeing well-connected communities. *arXiv:1810.08473 [physics]*, October 2018. URL http://arxiv.org/abs/1810.08473. arXiv: 1810.08473.

[44] Dmitry Kobak and George C. Linderman. UMAP does not preserve global structure any better than t-SNE when using the same initialization. *bioRxiv*, page 2019.12.19.877522, December 2019. doi: 10.1101/2019.12.19.877522. URL https://www.biorxiv.org/content/10.1101/2019.12.19.877522v1.

[45] Evan Z. Macosko, et al. Highly Parallel Genome-wide Expression Profiling of Individual Cells Using Nanoliter Droplets. *Cell*, 161(5):1202–1214, May 2015. ISSN 0092-8674, 1097-4172. doi: 10.1016/j.cell.2015.05.002. URL https://www.cell.com/cell/abstract/S0092-8674(15)00549-8.

[46] Jong-Eun Park, et al. Fast Batch Alignment of Single Cell Transcriptomes Unifies Multiple Mouse Cell Atlases into an Integrated Landscape. *bioRxiv*, page 397042, August 2018. doi: 10.1101/397042. URL https://www.biorxiv.org/content/early/2018/08/22/397042.

[47] Josip S. Herman, et al. FateID infers cell fate bias in multipotent progenitors from single-cell RNA-seq data. *Nature Methods*, 15(5):379–386, May 2018. ISSN 1548-7105. doi: 10.1038/nmeth.4662. URL https://www.nature.com/articles/nmeth.4662. Number: 5 Publisher: Nature Publishing Group.

[48] Lars Velten, et al. Human haematopoietic stem cell lineage commitment is a continuous process. *Nature Cell Biology*, 19(4):271–281, April 2017. ISSN 1476-4679. doi: 10.1038/ncb3493. URL https://www.nature.com/articles/ncb3493. Number: 4 Publisher: Nature Publishing Group.

[49] Brent A. Biddy, et al. Single-cell mapping of lineage and identity in direct reprogramming. *Nature*, 564(7735):219, December 2018. ISSN 1476-4687. doi: 10.1038/s41586-018-0744-4. URL https://www.nature.com/articles/s41586-018-0744-4.

[50] Aimée Bastidas-Ponce, et al. Cellular and molecular mechanisms coordinating pancreas development. *Development*, 144(16):2873–2888, August 2017. ISSN 0950-1991, 1477-9129. doi: 10.1242/dev.140756. URL https://dev.biologists.org/content/144/16/2873. Publisher: Oxford University Press for The Company of Biologists Limited Section: REVIEW.

[51] Patrick Collombat, et al. Opposing actions of Arx and Pax4 in endocrine pancreas development. *Genes & Development*, 17(20):2591–2603, October 2003. ISSN 0890-9369, 1549-5477. doi: 10.1101/gad.269003. URL http://genesdev.cshlp.org/content/17/20/2591. Company: Cold Spring Harbor Laboratory Press Distributor: Cold Spring Harbor Laboratory Press Institution: Cold Spring Harbor Laboratory Press Label: Cold Spring Harbor Laboratory Press Publisher: Cold Spring Harbor Lab.

[52] Aimée Bastidas-Ponce, et al. Foxa2 and Pdx1 cooperatively regulate postnatal maturation of pancreatic -cells. *Molecular Metabolism*, 6(6):524–534, June 2017. ISSN 2212-8778. doi: 10.1016/j.molmet.2017.03.007. URL http://www.sciencedirect.com/science/article/pii/S2212877817300510.

[53] J. Jonsson, et al. Insulin-promoter-factor 1 is required for pancreas development in mice. *Nature*, 371 (6498):606–609, 1994. doi: 10.1038/371606a0.

[54] D.A. Stoffers, et al. Pancreatic agenesis attributable to a single nucleotide deletion in the human IPF1 gene coding sequence. *Nature Genetics*, 15(1):106–110, 1997. doi: 10.1038/ng0197-106.

[55] Jia Zhang, et al. The diabetes gene Hhex maintains -cell differentiation and islet function. *Genes & Development*, 28(8):829–834, April 2014. ISSN 0890-9369. doi: 10.1101/gad.235499.113. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4003275/.

[56] C. Berthault, et al. Purification of pancreatic endocrine subsets reveals increased iron metabolism in beta-cells. *Molecular Metabolism*, page 101060, August 2020. ISSN 22128778. doi: 10.1016/j.molmet.2020.101060. URL https://linkinghub.elsevier.com/retrieve/pii/S2212877820301344.

[57] David S. Cram, et al. Differential mRNA display analysis of two related but functionally distinct rat insulinoma (RIN) cell lines: identification of CD24 and its expression in the developing pancreas. *Differentiation*, 64(4):237–246, May 1999. ISSN 03014681. doi: 10.1046/j.1432-0436.1999.6440237.x. URL https://linkinghub.elsevier.com/retrieve/pii/S0301468109603984.

[58] Lauren E. Byrnes, et al. Lineage dynamics of murine pancreatic development at single-cell resolution. *Nature Communications*, 9(1):3922, September 2018. ISSN 2041-1723. doi: 10.1038/s41467-018-06176-3. URL https://www.nature.com/articles/s41467-018-06176-3. Number: 1 Publisher: Nature Publishing Group.

[59] Nicole A. J. Krentz, et al. Single-Cell Transcriptome Profiling of Mouse and hESC-Derived Pancreatic Progenitors. *Stem Cell Reports*, 11(6):1551–1564, December 2018. ISSN 2213-6711. doi: 10.1016/j.stemcr.2018.11.008. URL https://www.cell.com/stem-cell-reports/abstract/S2213-6711(18)30476-4. Publisher: Elsevier.

[60] Fabian Pedregosa. memory-profiler: A module for monitoring memory usage of a python program. URL https://github.com/pythonprofilers/memory_profiler.

[61] Thomas Quinn. peakRAM: Monitor the Total and Peak RAM Used by an Expression or Function, January 2017. URL https://CRAN.R-project.org/package=peakRAM.

[62] Maslinda Musa, et al. Differential localisation of BPIFA1 (SPLUNC1) and BPIFB1 (LPLUNC1) in the nasal and oral cavities of mice. *Cell and Tissue Research*, 350(3):455–464, December 2012. ISSN 1432-0878. doi: 10.1007/s00441-012-1490-9.

Figures

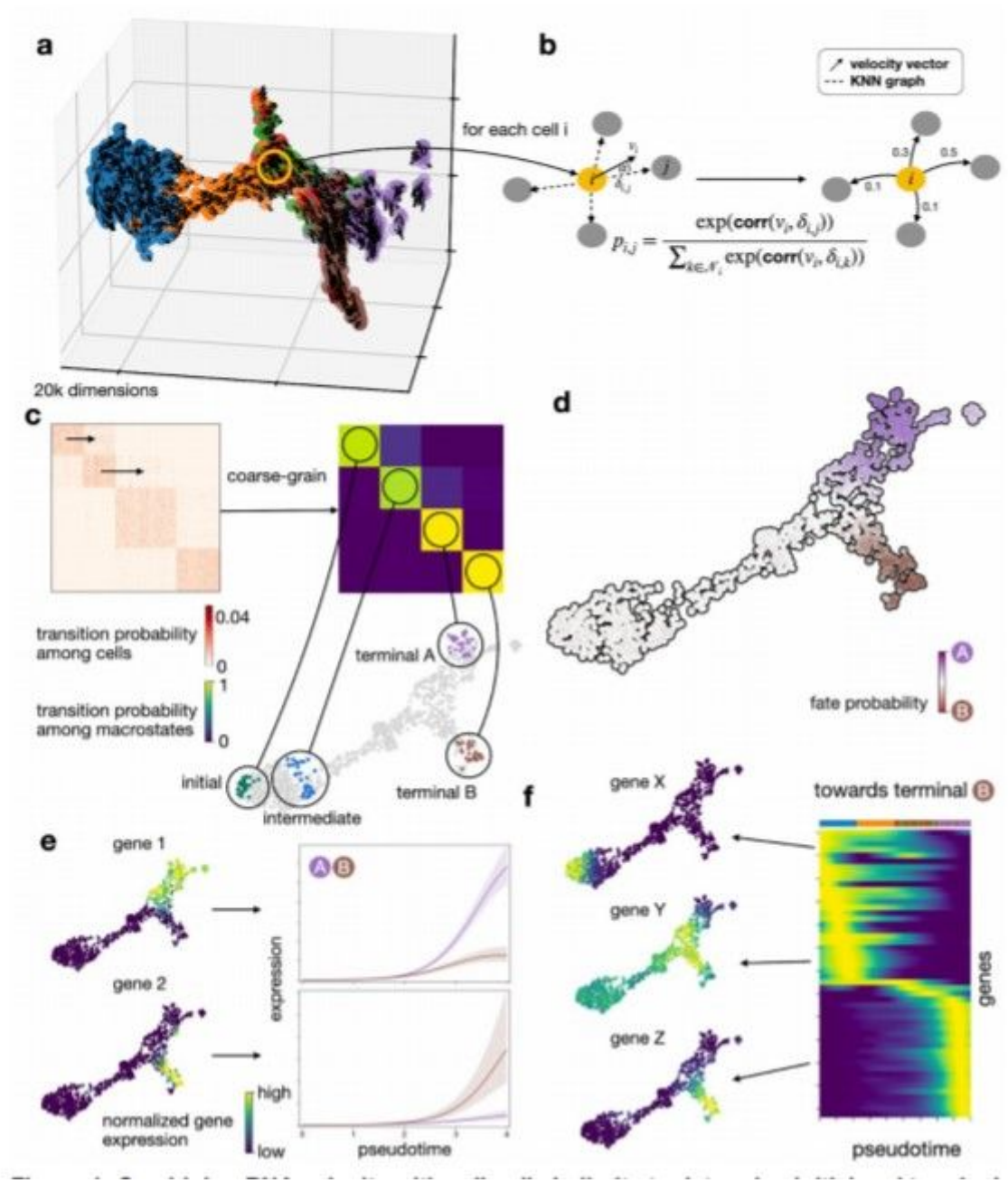

Figure 1

Combining RNA velocity with cell-cell similarity to determine initial and terminal states and compute a global map of cellular fate potential.

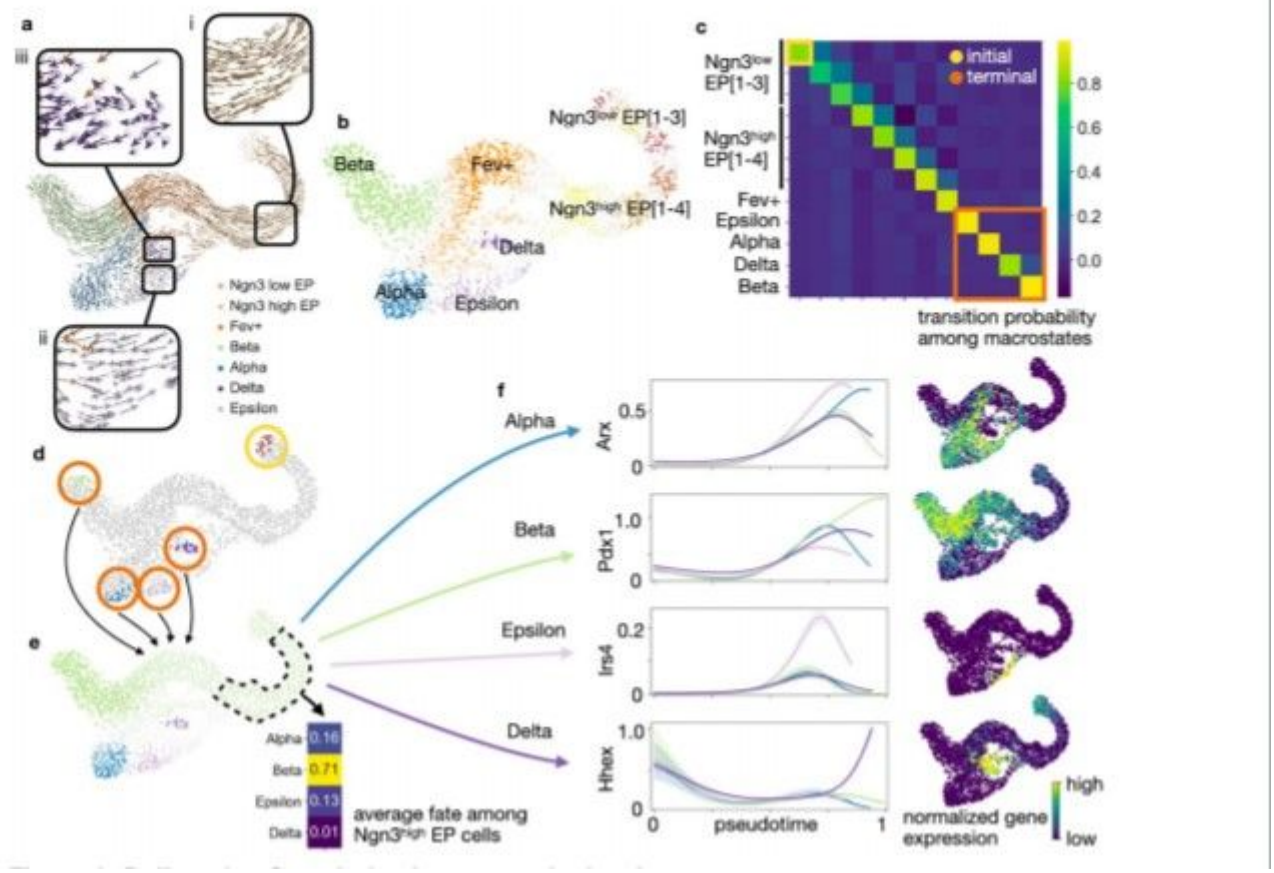

Figure 2

Delineating fate choice in pancreatic development

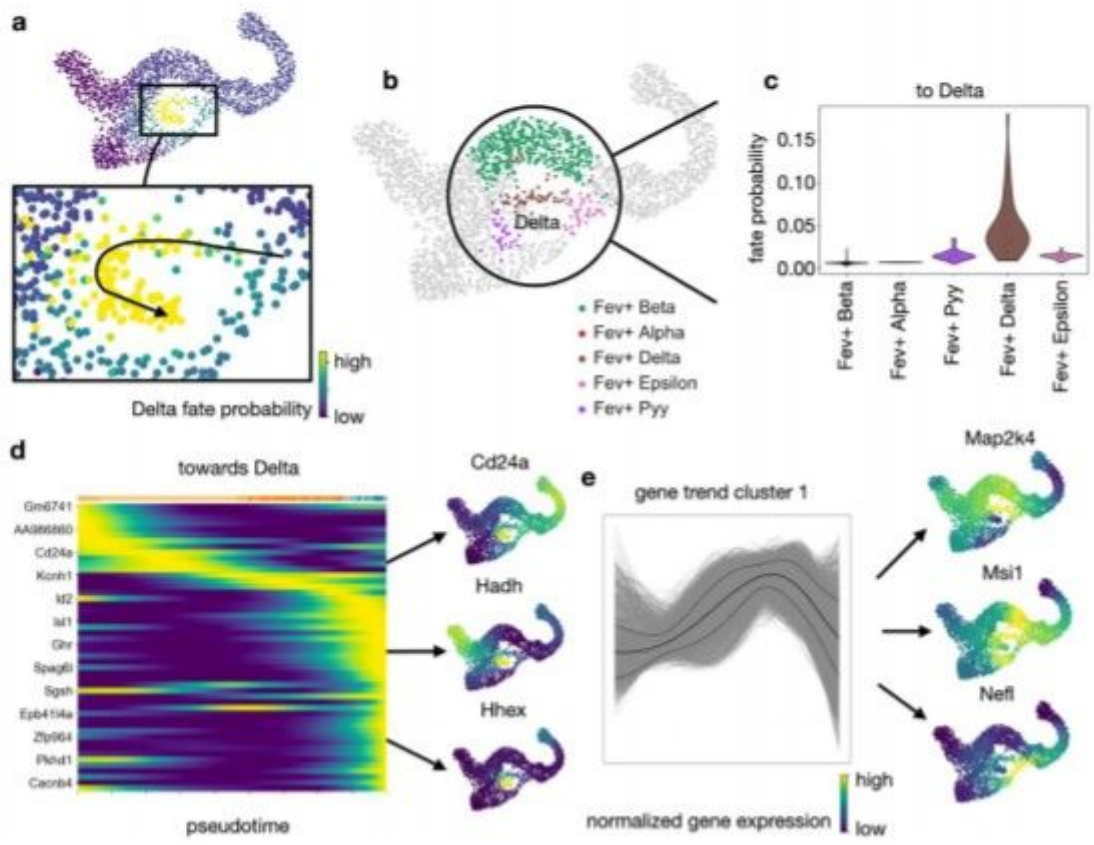

Figure 3

Zooming into the delta state to elucidate differentiation paths

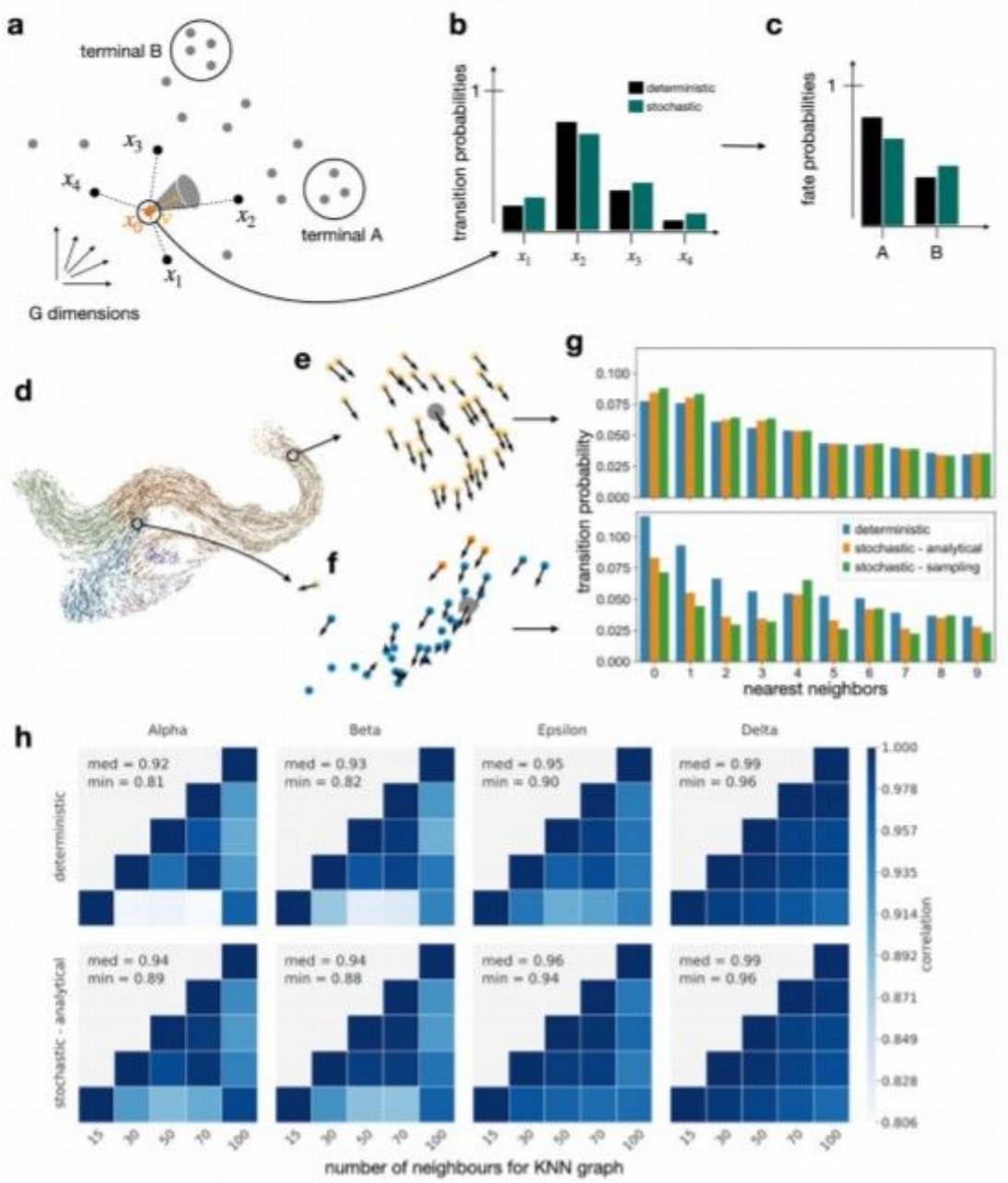

Figure 4

Uncertainty propagation adjusts for noise in RNA velocity vectors

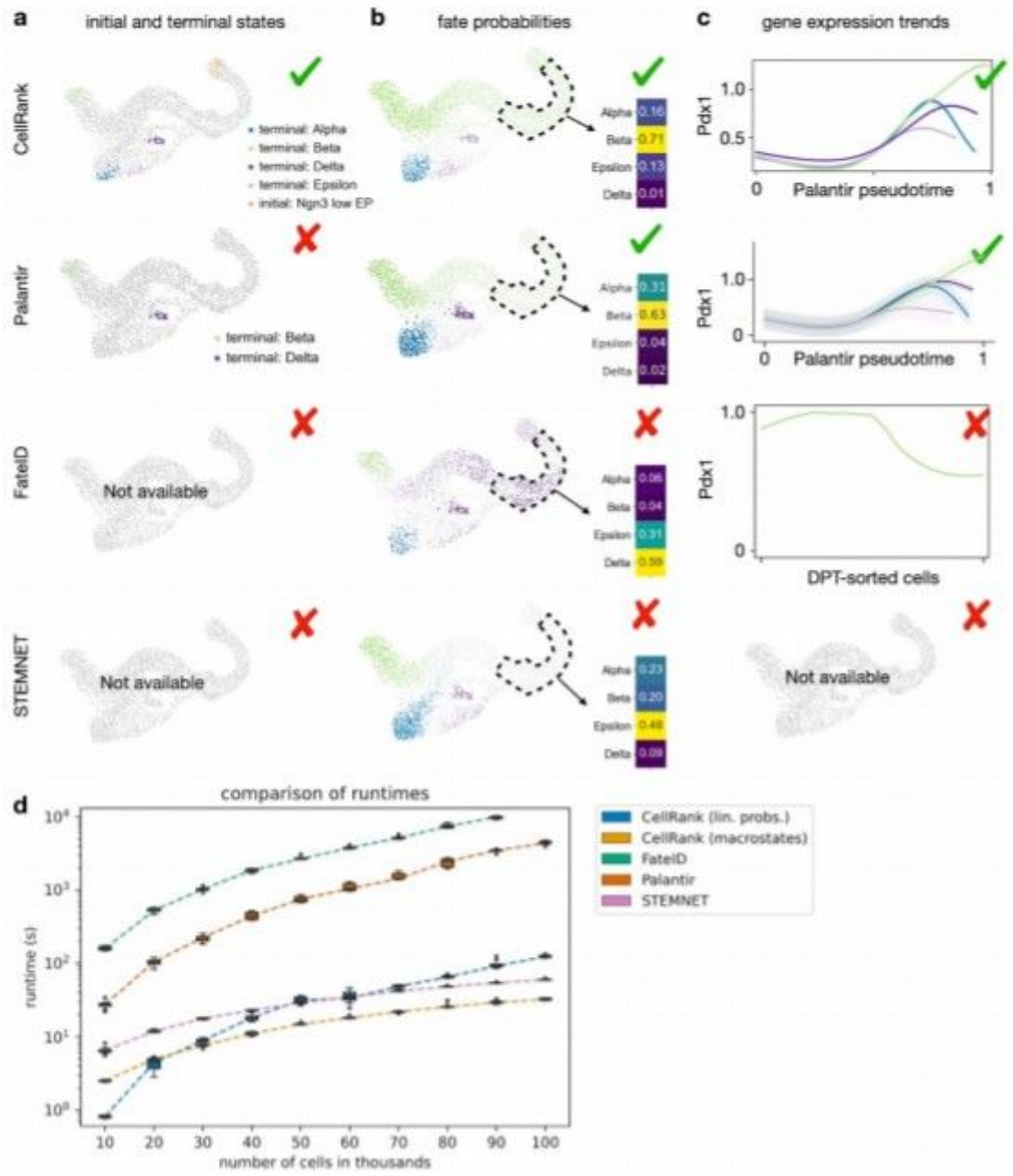

Figure 5

CellRank outperforms methods that do not include RNA velocity

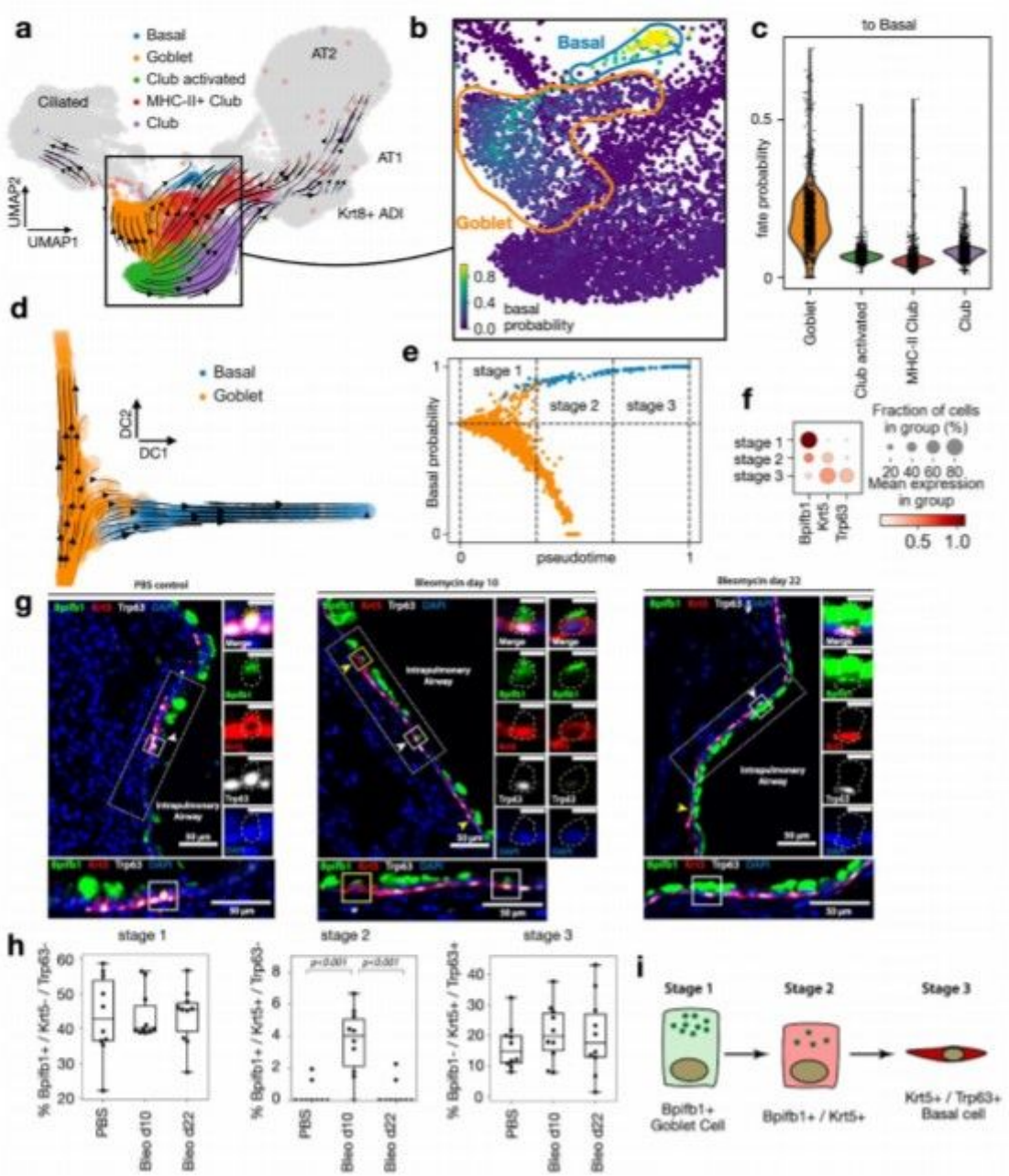

Figure 6

Cellrank predicts a novel differentiation trajectory in Murine lung regeneration

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- 20201019cellranksupplementarytables.pdf
- 20201019cellranksupplementaryfigures.pdf