Supplementary Note 1

Tomáš Kalina[1], David Novák[2,3], Naděžda Brdičková[1], and Jan Stuchlý[1]

[1]Department of Paediatric Haematology and Oncology,
2nd Faculty of Medicine, Charles University. Prague, Czech Republic.
[2]Data Mining and Modeling for Biomedicine, VIB Center for Inflammation Research, Ghent, Belgium.
[3]Department of Applied Mathematics, Computer Science and Statistics, Ghent University, Ghent, Belgium.

March 14, 2023

1 Datasets

1.1 Dyntoy dataset

For the purpose of basic illustrations we created an artificial dataset using *dyntoy* R package (Cannoodt and Saelens (2022)) as follows:

```
devtools::install_github('dynverse/dyntoy')
library(dyntoy)
n_events <- 10000
n_features <- 3896
set.seed(12345)
d <- dyntoy::generate_dataset(
  id           = 'tviblindi_dyntoy_test',
  model        = 'connected',
  num_features = n_features,
  num_cells    = n_events
)
```

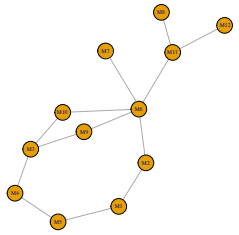
and used first 25 principal component for the analysis (see Figure 1a for the topology of this dataset). To further underline some issues in *trajectory inference*, we modified this dataset by up-sampling the population M10 - we iterated over the points and added random convex combinations of their neighbors in the k-NN graph. This resulted in a new dataset with 35795 points where 26532 points lie in the population M10 - we will refer to this dataset as *up-sampled dyntoy dataset* (see Figure 1d,e).

1.2 Mice thymus scRNAseq

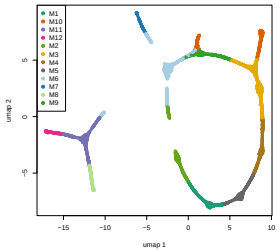
1.3 Human thymus & PBMC

2 Dimensionality reduction

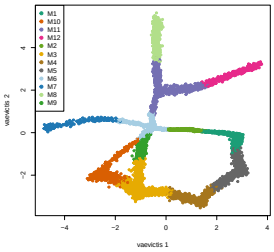
To meet the requirements for data visualization in TI problems i.e. the continuity, robustness against the difference in size of cell populations and straightforward extrapolation on new datasets, we implemented a variational autoencoder based on ideas in Szubert et al. (2019); Ding et al. (2018) called *vaevictis*. The objective function corresponds to weighted sum of reconstruction error, variational regularization, t-sne regularization and triplet loss-function. For additional speedup (which make the time complexity virtually independent of dataset size) and improved robustness against differences in population size the dataset is by default first clustered by k-means clustering and a fixed number of representatives are sampled from each cluster. Method implementation: https://github.com/stuchly/vaevictis.



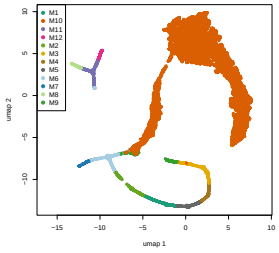
(a) Topology of dyntoy data



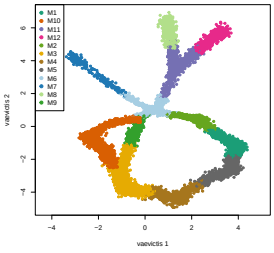
(b) Umap on dyntoy data



(c) Vaevictis on dyntoy data



(d) Umap on up-sampled data



(e) Vaevictis on up-sampled data

Figure 1: Feature preservation in DR methods

On Figure 1 we illustrate the performance of *vaevictis* on *(up-sampled) dyntoy dataset* (subsection 1.1) as compared to *umap* the overall structure of the data (panel a) is preserved

by *vaevictis* (panel c) even for data with numerically unbalanced populations (panel d;the population M10 was enlarged to contain $\sim 74\%$ of all cells, see subsection 1.1). On the other hand *umap* preserves moderate distance relationships comparatively worse (panel b and d).

3 Pseudotime estimation & random walks simulation

Let $G = (V, E)$ be k-NN graph (or any suitable connected graph over the data), where V and E stand for vertices (cells) and edges of the graph respectively. We have the corresponding *distance matrix* S

$$S_{ij} = \begin{cases} d(v_i, v_j) & (v_i, v_j) \in E \\ \infty & \text{otherwise} \end{cases} \tag{1}$$

where d is a distance function (in our implementation we use Euclidean distance). For the modeling of diffusion on G we also consider *transition matrix* T

$$T_{ij} = \begin{cases} f(S_{ij}) & (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where f is a suitable kernel function. A sensible choice is the Gaussian kernel

$$f(x) = e^{-x^2/\varepsilon},$$

ε is determined from the distribution of vertices as follows: For an edge $(v_i, v_j) \in E$ $\varepsilon = \text{median}\{d(v_i, v_k)^2, (v_i, v_k) \in E\}$. For convenience we define *transition probability matrix*

P

$$P = D^{-1}T \tag{3}$$

where D is a diagonal matrix of rowsums of T.

We will calculate the *pseudotime* of each vertex of G in the terms of expected length of random walks.

Definition 1 (Random walk)**.** *We define a random walk (of possibly infinite length) on $G = (V, E)$ as a series $v^0, v^1, ...$ of vertices in G. Each v^{i+1} is chosen from the set $\{v_j : \{v_l, v_j\} \in E, v_l = v^i\}$ with probability $p_{v_j} = P_{lj}$ (P defined in 3).*

If we fix the starting point v_0 of random walks we can define the *pseudotime* as one of following quantities

Definition 2 (Pseudotime)**.** *Let $v_0 \in V$ be fixed. we define pseudotime $\tau_{v_0}(v)$ for each $v \in V$ as either* **Expected hitting time** *- the expected length of a random walk sequence starting in v_0 before it reaches v or as* **Expected hitting distance** *- the expected distance traveled by a random walk starting in v_0 before it reaches v. For both definitions holds $\tau_{v_0}(v_0) = 0$.*

The *expected hitting time* can be calculated (for $v_j \neq v_0$) as follows (e.g. Förster et al. (2022))

$$\tau_{v_0}(v_j) = \sum_{v_i : \{v_i, v_j\} \in E} P_{ij}(\tau_{v_0}(v_i) + 1) \text{ if } v_j \neq v_0,\ \tau_{v_0}(v_0) = 0$$

The generalization to *expected hitting distance* is straightforward

$$\tau_{v_0}(v_j) = \sum_{v_i : \{v_i, v_j\} \in E} P_{ij}(\tau_{v_0}(v_i) + S_{ij}) \text{ if } v_j \neq v_0,\ \tau_{v_0}(v_0) = 0$$

Rewriting into matrix form (with components corresponding to v_0 dropped) we get

$$\boldsymbol{\tau} = P\boldsymbol{\tau} + (P \odot S)\mathbf{1}$$

where $\odot$ denotes the Hadamard (element-wise) product and $\mathbf{1}$ is a vector of ones. We can solve for τ the equation

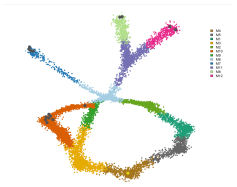
$$(I - P)\boldsymbol{\tau} = (P \odot S)\mathbf{1} \tag{4}$$

where I denotes the identity matrix. Note that if T is symmetric (which is by default enforced by setting $T_{ij} = max(T_{ij}, T_{ji})$) the equation 4 can be rewritten into the form $D^{1/2}(I-P)D^{-1/2}D^{1/2}\tau = D^{1/2}(P \odot S)\mathbf{1}$, where $D^{1/2}(I-P)D^{-1/2}$ is a symmetric matrix. The equation 4 is a sparse system which can be efficiently solved by iterative methods such as (bi)conjugate gradient method even for graphs with millions of vertices. This allows to keep the resolution of pseudotime on single-cell level.

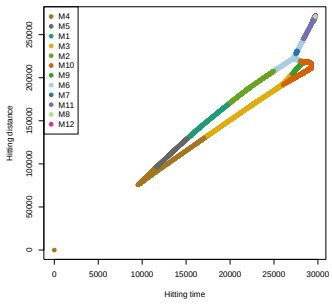
Once the *pseudotime* is estimated, we can use it to simulate finite random walks. We do it by creating directed acyclic graph (DAG) $G' = (V, E')$, where $E' = \{(v_i, v_j) \in E,\ \tau(v_j) > \tau(v_i)\}$. Random walks on a DAG are necessarily finite and their final vertices are suitable candidates of developmental endpoints. We simulate large number of walks on G' and record the endpoints for further analysis (e.g. on Figure 2 in gray) where we study the possible trajectories (represented by sets of spatially coherent random walks) by which the endpoints can be reached.

The main motivation for introducing the *expected hitting distance* was the need of a robust *pseudotime* estimation even in the case of significant accumulation of cells in some stages of development. The *expected hitting time* tends to rise quickly within compact numerically large populations which would yield unwanted points of attraction in further

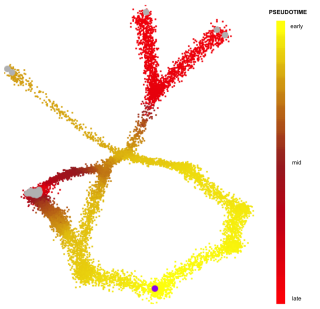
analysis (see below). To illustrate this issue we took the *up-sampled dyntoy dataset* (see subsection 1.1 and Figure 2). We set as v_0 the centroid of population M4. If the *pseudotime* is calculated as *expected hitting time* the population M10 forms a sink which blocks any random walk entering it (see the color gradient on Figure 2c). On the other hand the *pseudotime* based on *expected hitting distance* allows random walks to pass through the population M10 (Figure 2d). Indeed the Figure 2b shows that passing through the population M10 means to go back in time when using the *expected hitting time* formulation.



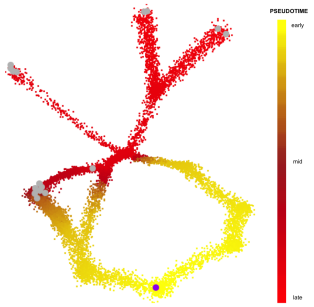
(a) Dyntoy data with marked populations



(b) Hitting time vs hitting distance



(c) Pseudotime - hitting time



(d) Pseudotime - hitting distance

Figure 2: Pseudotime calculation strategies

3.1 On the numerical aspects of equation 4

One of the crucial advantages of describing the *pseudotime* as a solution of a linear system $Ax = b$ is the possibility to measure the accuracy of our solution as the relative error $\frac{\|Ax-b\|}{\|b\|}$. Although the iterative methods can solve the system (provided the solution exists) to any precision (limited by machine precision of floating point arithmetic), in practice the desired precision might not be possible to reach (e.g. because of ill-conditioned system).

Although it happened rarely in our experiments, it is an issue which needs attention. In most cases it could be rectified by changing the kernel function (e.g. Gaussian kernel seems to work better than exponential kernel on data from classical flow cytometry) or adjusting the transformation of the data ($asinh(.)$ transformation is usually a sensible choice).

To summarize, the reported error offers an important diagnostic tool of *pseudotime* estimation.

4 Topological clustering of random walks

Our approach to the random walks clustering is based on the idea presented in Pokorny et al. (2014). Detailed treatment of relevant mathematical tools can be found in Hatcher (2000); Edelsbrunner and Harer (2022).

4.1 Basic notions of simplicial homology

For convenience we provide in this subsection definitions of basic terms we use in the rest of this technical note.

Definition 3 (p-simplex)**.** *We define a* p**-simplex** *as a convex hull of* $p+1$ *affinely independent points.*

A p*-simplex* σ *is a point for* $p = 0$*, a line segment for* $p = 1$*, a triangle for* $p = 2$ *and a*

tetrahedron for $p = 3$ etc. We use the notation $\sigma = [v_0, v_2, \ldots, v_p]$ where v_i are vertices. The convex hull of a nonempty subset of vertices is called a **face** *of the simplex.*

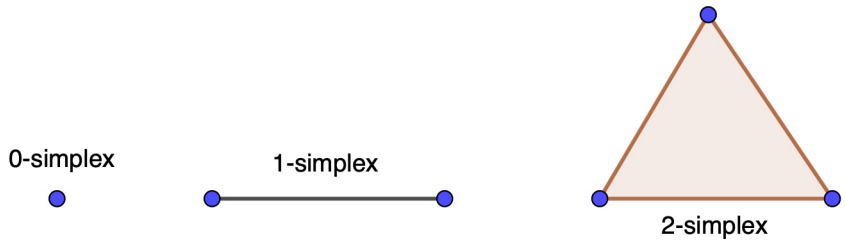


Figure 3: Simplices

Definition 4 (Simplicial complex)**.** *A* **simplicial complex** *K is a set of simplices that satisfies the following conditions:*

1. *Every face of a simplex from K is also in K.*
2. *The non-empty intersection of any two simplices $\sigma_1, \sigma_2 \in K$ is a face of both σ_1 and σ_2.*

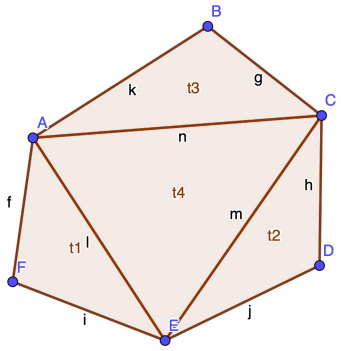


Figure 4: Simplicial complex

Definition 5 (p-chain group)**.** *For a simplicial complex K, the group $C_p(K)$ of* **p-chains** *of*

K is defined as a formal linear combination of p-simplices:

$$C_p(K) = \left\{ \sum_i m_i \sigma_i | m_i \in \mathbb{Z}/2\mathbb{Z} \right\},$$

where σ_i are p-simplices of K.

Note *that the coefficients $m_i \in \mathbb{Z}/2\mathbb{Z}$. In general the coefficients may be from any ring. This choice simplifies many things (e.g. no issues with orientation as $-1 = 1$) but we lose some descriptive power.*

Definition 6 (Boundary)**.** *The* **boundary homomorphism** *$\partial_p : C_p(K) \to C_{p-1}(K)$ is defined simplex-wise for a simplex $\sigma = [v_0, v_2, ..., v_p]$ as*

$$\partial\sigma = \sum_{j=0}^{p} [v_0, \ldots, v_{j-1}, \hat{v}_j, v_{j+1}, \ldots, v_p],$$

where $\hat{v}_j$ indicates deletion of vertex v_j.

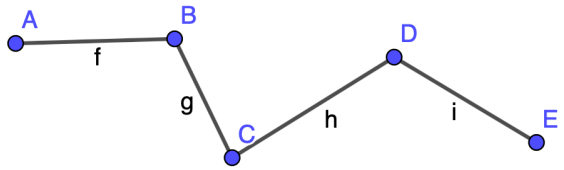


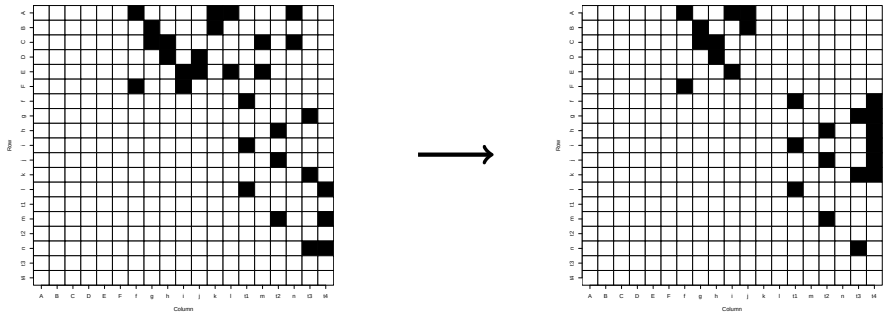
Figure 5: 1-chain $c = f + g + h + i = [A, B] + [B, C] + [C, D] + [D, E]$

$$\begin{aligned} \partial c = \partial f + \partial g + \partial h + \partial i &= \partial[A, B] + \partial[B, C] + \partial[C, D] + \partial[D, E] \\ &= A + B + B + C + C + D + D + E = A + E \end{aligned}$$

(coefficients in $\mathbb{Z}/2\mathbb{Z}$)

The boundary operator can be expressed as a matrix

$$\partial_{i,j} = \begin{cases} 1, & \text{if } \sigma_i \text{ is a co-dimension 1 face of } \sigma_j \\ 0, & \text{otherwise} \end{cases}$$



(a) Boundary matrix (b) Reduced boundary matrix

Figure 6: Boundary matrix of simplicial complex from figure 4, 1s are marked as black squares

Definition 7 (Reduced (boundary) matrix)**.** *Let R be a matrix with elements in $\mathbb{Z}/2\mathbb{Z}$ and $Low(j)$ be the row index of the lowest* 1 *in the column j of the matrix R and $Low(j)$ undefined on zero columns. We say that matrix R is* **reduced** *if for any two non-zero columns i, j, $i \neq j$ $Low(i) \neq Low(j)$. See Figure 6.*

Matrix reduction can be performed by efficient algorithms and is essential for calculation of persistence (Definition 11). Matrix reduction corresponds to left-to-right column addition of columns representing boundaries of same dimension, hence the columns of *reduced boundary matrix* correspond to boundaries of a simplicial complexes (sums of boundaries of some p-simplices).

Definition 8 (Boundaries & Cycles)**.** *We call*

*p-***boundaries***: the group $B_p = Im\ \partial_{p+1}$*

*p-***cycles***: the group $Z_p = Ker\ \partial_p$*

*p-***th homology group***[1]: $H_p = Z_p/B_p$*

[1]Note that by fundamental lemma of homology $\partial_p \partial_{p+1} c = 0$ holds for any $p+1$-chain c and consequently $B_p \subset Z_p$ (see Figure 7)

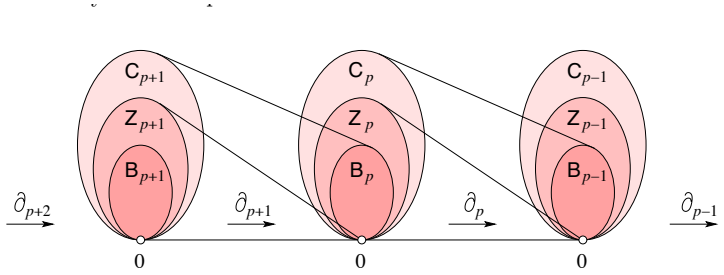


Figure IV.1: The chain complex consisting of a linear sequence of chain, cycle, and boundary groups connected by homomorphisms.

Figure 7: Edelsbrunner & Harer - Computational Topology: An Introduction

Definition 9 (Filtration)**.** *The* **filtration** *is a sequence of simplicial complexes that are ordered by inclusion. Hence, if K_1 and K_2 are complexes where K_1 appears before K_2 in the filtration, then $K_1 \leq K_2 : K_1$ is a sub-complex of K_2.*
By the **filtration of a simplicial complex** *K we mean a sequence*

$$\emptyset = K_0 \subseteq K_1 \subseteq K_2 \subseteq \ldots \subseteq K_n = K$$

These inclusions induce a sequence of homology groups connected by homomorphisms

$$f_p^{i,j} : H_p(K_i) \to H_p(K_j),\ i \leq j$$

The filtration thus gives a sequence of homology groups

$$0 = H_p(K_0) \to H_p(K_1) \to H_p(K_2) \to \ldots \to H_p(K_n) = H_p(K).$$

Definition 10 (Filtration induced by monotonic function)**.** *Let K be a simplicial complex and $f : K \to \mathbb{R}$ a monotonic function on K, that is $f(\sigma) \leq f(\tau)$ whenever σ is a face τ.*
Let $a_1 < a_2 \ldots < a_n$ be the values of function f on the simplices in K and $a_0 = -\infty$, then

the sequence $K_i = f^{-1}(-\infty, a_i]$ *is a* **filtration of** K **induced by** f. *We call* a_i **filtration values**.

Definition 11 (Persistence and essential classes)**.** *Given a filtration,* p**-th persistent homology groups** *are defined as* $H_p^{i,j} := Imf_p^{i,j}$, *for* $0 \leq i \leq j \leq n$. *(i.e.* $H_p^{i,j} = Z_p(K_i)/(B_p(K_j) \cap Z_p(K_i))$*).*

We say that a homology class $\gamma \in H_p(K_i)$ *is* **born** *at* K_i *if* $\gamma \notin H_p^{i-1,i}$. *Furthermore we say, that a class* γ *born in* K_i **dies** *entering* K_j *if* $f_p^{i,j-1}(\gamma) \notin H_p^{i-1,j-1}$ *but* $f_p^{i,j}(\gamma) \in H_p^{i-1,j}$.

If the filtration is induced by a monotonic function we define the **persistence** *of class* γ *which was born at* K_i *and died at* K_j *as* $pers(\gamma) = a_j - a_i$. *Sometimes it is convenient to consider* **index persistence** $= j - i$.
Throughout the text we will refer to the subcomplexes or corresponding filtration values where a homology class is born or where it dies as **birth** *and* **death** *of the class.*

We say that a p*-cycle* c *represents an* **essential class** *in the simplicial complex* K *iff there there is no* $(p+1)$*-chain* $k' \in C_{p+1}(K)$ *such that* $c = \partial k'$. *In other words the essential classes are persistent homology classes which never die.*

If we consider a *reduced boundary matrix* with columns and row ordered by filtration then every nonzero column j corresponds to a *death* of a homology class. Moreover this class was *born* at $K_{\mathrm{Low}(j)}$, thus the *index persistence* of this homology class is $j - \mathrm{Low}(j)$.

4.2 Triangulation of points cloud & random walks

The triangulation of high dimensional point cloud is a computationally intensive task. The complexity of building triangulations used in lower dimensional problems such as *Delauney*

triangulation grows exponentially with the dimension, while other triangulations better suited for high-dimensional problems like *Vietoris-Rips complex* can become very large if we need sufficient cover of the space. We decided to employ the *witness complex* (Silva and Carlsson (2004)) implemented in Kachanovich (2015). *Witness complex* is built from a smaller set of landmark points and uses the original point cloud to witness the existence of connections between points. In our implementation we first cluster the points in large number of clusters (by default 625) by k-means clustering and use the centroids of the clusters as landmark points. Obviously the vertices of *witness complex* differ from the vertices in G' where the random walks were simulated. We need to triangulate the random walks as well. The procedure is straightforward, we simply contract the vertices to the corresponding cluster and remove loops from contracted random walks. Thus, after the triangulation step, each random walk corresponds to a connected 1-chain in the *witness complex*.

The witness complex can be built up to a preset dimension of simplices. Since we need only 2-simplices to classify 1-chains we limit the dimension to 2 and effectively create a 2-skeleton. The filtration values of simplices are then calculated as for alpha complex filtration (Rouvreau (2015)).

4.2.1 Data denoising

Technical noise is always present in single-cell data. The presence of unwanted data points in sparse regions which are used for characterization of the shape of our point cloud can make the use of persistent homology less effective at describing our data. In order to reduce this noise and help capture the topology correctly, we apply the following denoising technique, suggested in Carlsson et al. (2008).

Taking advantage of having built a k-NN graph previously, we replace the coordinates

of each data point with the averaged coordinates of some l of its nearest neighbors. The denoised version of our expression matrix is used to make the construction of our witness complex.

4.3 Cycle representation

In order to classify random walks with respect to homological classes we need a suitable representation of their dissimilarity. The basic idea of this approach is outlined in Pokorny et al. (2014). We however improve on the cited results by allowing *essential classes* and consequently the classification of random walks with different endpoints (e.g. developmental branching).

Every random walk can be understood as a connected 1-chain with at least one vertex in it's boundary (the cell-of-origin vertex v_0) common with all other random walks. Consider first a set of 1-chains $\{c^0, c^1, \ldots, c^N\}$ with common boundary (i.e. origins and endpoints of all corresponding random walks are the same). We can now fix a reference 1-chain (e.g. c^0) and consider 1-*cycles* $\{c^0+c^1, c^0+c^2, \ldots, c^0+c^N\}$. If all cycles in this set are boundaries of some 2-chain, they can be uniquely expressed as linear combinations (over $\mathbb{Z}/2\mathbb{Z}$) of columns of the *reduced boundary matrix* (as described in Pokorny et al. (2014)). However if some of these *cycles* represent an *essential class* we need a more elaborate solution implemented in Algorithm 1. If during the calculation of the representation we encounter a *cycle* which is not a *boundary* we simply form an auxiliary *cycle* representing the "essential part" and add it as another column into the *reduced boundary matrix*. The filtration value corresponding to the *death* of this class is ∞, but for the purpose of visualization on persistent diagram we assign to all *essential classes* arbitrary values so that they have the same *death/birth* ratio and *death* filtration value both higher than any non-essential classes.

If we fix a reference 1-chain c^0, we can uniquely represent this way any 1-chain c^i with

the same boundary by expressing the *cycle* c^0+c^i by a set of columns of (possibly updated) *reduced boundary matrix*. Note that $c^0 + c^0$ is an empty 1-chain. This representation will be used in subsection 4.4 to hierarchically cluster random walks.

It remains to discuss the case of classification of random walks with different endpoints. Two options are implemented. First option is to simply choose the endpoint with highest *pseudotime*, set it as the endpoint of all random walks in question, and connect the other endpoint(s) to it by the shortest path within the simplicial complex. This approach is convenient in the case of closely clustered endpoints representing the same developmental fate. Second option is to add 1-simplex between selected endpoints. Since there were no 1-simplices connecting these endpoints (otherwise random walks in question would end in the endpoint with the higher *pseudotime*), this will effectively create (a) new *essential class(es)*. Random walks differing by an *essential class* will be split into major branches at the root of the dendrogram (see subsection 4.4).

We close this subsection by describing the algorithms in detail and with proof of the correctness. Since we use as the coefficients ring the field $\mathbb{Z}_2$ we can understand simplicial complexes (and consequently p-chains) as sets of simplices. To simplify further exposition we introduce following notation and arithmetic:

Definition 12 (Notation and arithmetic of p-chains & Low^{-1})**.** *Given a simplicial complex* $K = \{\sigma_1, \sigma_2, \ldots, \sigma_N\}$ *and a* p*-chain* $c = \{\sigma_{i_1}, \sigma_{i_2}, \ldots, \sigma_{i_k}\}$*, we denote the* **index representation** c_I *of* c *the tuple* $c_I = (i_1, i_2, \ldots, i_K)$*.*
For any two p*-chains* a_I*,* b_I *we define (in agreement with the arithmetic over* $\mathbb{Z}_2$*)*

$$a_I + b_I = a_I \triangle b_I \text{ (the symmetric difference of the 2 sets).}$$

By a slight abuse of notation, we will define the **index representation of a set of** p**-chains**

$C = \{c^1, \ldots, c^k\}$ *as* $C_I := (c^1 + c^2 + \ldots + c^k)_I$.

We will also assume that the indices in the index representation tuple are always ordered with respect to the filtration in descending order i.e. for any p-chain c, $c_I[1]$ *has the highest filtration value among all simplices forming the p-chain.*

Let us further define for the reduced boundary matrix R the map Low^{-1}*:*

$Low^{-1}(i) = j$ *if* i *is the highest row index of a non-zero element in column* j *in matrix* R *and* $Low^{-1}(i) = -1$ *if there is no such column.*

Algorithm 1 One cycle representation

Input: c_I cycle, R reduced boundary matrix with corresponding Low^{-1} function

Output: r representation (tuple of column indices) of the cycle c_I as columns of (updated) boundary matrix R, updated boundary matrix R

```
procedure GET_REPRESENTATION_OF_CYCLE(c_I,R)
    r ← ∅
    e_I ← ∅
    b_I ← ∅
    N ← number of columns of R
    while c_I ≠ ∅ do
        j ← Low^{-1}(c_I[1])
        if j>0 then
            r ← r ∪ j
            c_I ← c_I + R[,j]_I
            b_I ← b_I + R[,j]_I
            e_I ← e_I
        else
            e_I ← e_I + c_I[1]
            c_I ← c_I + c_I[1]
            r ← r
        end if
    end while
    if e_I ≠ ∅ then
        R ← [R, e_I] (adding a column with 1s at positions e_I and 0s elsewhere
        Low^{-1} ← update Low^{-1} function with respect to the new R.
        r ← r ∪ (N + 1)
    end if
    return r, R
end procedure
```

Algorithm 2 Cycles representation

Input: $C = \{c_I^1, c_I^2, \ldots, c_I^L\}$ set of cycles, R reduced boundary matrix with corresponding Low^{-1} function

Output: Representation $= \{r^1, r^2, \ldots, r^L\}$ representations (tuples of column indices of (updated) boundary matrix R) of cycles C

```
for i ∈ 1...L do
    r^i, R ← GET_REPRESENTATION_OF_CYCLE(c_I^i,R)
end for
return Representation
```

Proposition 4.1: Correctness of algorithms 1 and 2

i) Algorithm 1 finds unique representation for any p-cycle c_I in maximum of $\max(c_I)$ steps in the form $c_I = \hat{b}_I + \hat{e}_I$, where $\hat{b}_I$ is a p-boundary and $\hat{e}_I$ is a p-cycle representing essential class(es).

ii) Matrix R is updated in the i-th step of algorithm 2 iff $\forall j : 0 \leq j < i$ the cycle $c_I^i + c_I^j$ represents an essential class ($c_I^0 := \emptyset$).

Proof. Let us denote R_0 the original reduced boundary matrix and N_0 number of columns of R_0. Notice that for any e_I constructed by algorithm 1 there was no column with lowest non-zero element at $\max(e_I)$ before the update, hence the matrix R is always *reduced* and the columns remain linearly independent after the update(s). Consequently every element in their span has a unique representation.

(i) Since $c_I[1] = \max(c_I)$ and if $j = \mathrm{Low}^{-1}(c_I[1]) > 0$ then $\max(R[,j]_I) = \max(c_I)$, the value $\max(c_I)$ is reduced at least by 1 in every iteration.

By construction $\emptyset = c_I + b_I + e_I$, hence e_I constructed by algorithm 1 is a cycle. If $e_I \neq \emptyset$, the matrix R is updated by a column containing a cycle representing an essential class otherwise all cycles not representing an essential class can be expressed as a unique sum of boundaries and hence e_I would be $\emptyset$.

Consider representation r corresponding to the cycle c_I. Let us decompose r into disjoint sets $r = r_b \cup r_e$, such that $\min(r_e) > N_0$, $\max(r_b) \leq N_0$ and define $\hat{e}_I := R[,r_e]_I$, $\hat{b}_I := R[,r_b]_I$.

(ii) If there exists c_I^j, $0 \leq j < i$ such that the cycle $c_I^i + c_I^j$ does not represent an essential class, then by definition there exists a p-boundary b such that $c_I^i = c_I^j + b$. Since c_I^j has a representation within R and b can be expressed as sum of columns of R_0, no matrix update is necessary. Conversely if $\forall 0 \leq j < i$ the cycle $c_I^i + c_I^j$ represents an essential class, then

c_I^i cannot be expressed in the terms of the columns of matrix R and e_I constructed by algorithm 1 is non-empty. □

4.4 Topological hierarchical clustering of random walks

Consider a set of L random walks and corresponding 1-chains $\{c^0, c^1, c^2, \ldots, c^L\}$. Let c^0 is selected as reference 1-chain (thus having trivial representation) and the representations obtained by algorithm 2 be $\{\emptyset, r^1, r^2, \ldots, r^L\}$. Each r^i corresponds to a set of homological classes each with its own *birth* and *death* filtration values. If we set a threshold on *death* value and delete from the representation all homological classes which *died* before this point, the natural hierarchical clustering stems from the following idea. Initially each leaf of the dendrogram contains all random walks with the same representation, then every time a homological class dies as we are increasing the threshold two leaves/branches of the dendrogram may merge. As soon as all[2] homological classes *die*, the dendrogram is complete.

In practice we proceed differently. We first choose on the interactive persistence diagram homology classes we deem significant based on their persistence. Then we order the homology classes present in any representation in decreasing order with respect to their *death* value. Finally we split the random walks into two branches[3] based on the presence/absence of the homology class with highest *death* value in their representation, drop this homology class and re-iterate the process on each of the two subgroups.

[2]In practice we threshold on column indices of corresponding (updated) *reduced boundary matrix*. Thus even *essential classes die* after a finite number of steps

[3]If one branch is empty, we drop the class and re-iterate

5 GUI

An essential part of *tviblindi* is graphical user interface (GUI) which enables the on-demand aggregation of random walks and real-time resolution adjustments. In this section we describe main features of the GUI and the analysis process. See also supplementary videos. In the text we will refer to the annotated screenshots of the GUI on pages 27 and 28 of this note by numbers in circles (1) and squares [1] respectively. Please consult the section 6 or the documentation of *tviblindi* package for more details about the function mentioned in text (the function names are typeset as `verbatim`). Note that buttons provide tooltip help.

The user usually starts with adjustment of visualization and model selection. As every run of the function `DimRed` adds a 2D layout to *tviblindi* object we can choose the most appropriate visualization in the drop down menu (1). The menu (2) allows the user to choose the pseudotime and random walks model (different choice of cell-of-origin, different parameter choice of *pseudotime* calculation etc). As the visualization is optimized for large datasets (millions of events) the button (9) can increase the points size on scatter-plots. The labels (7) serve mainly for visualization of population on the plot invoked by (8) and for the tracking of population frequencies along the trajectory ((21) second pane).

This first step of the analysis is the choice of endpoints (3). The users selects the black dots by rectangular selection at once or iteratively by pressing (4) after each selection - The selected endpoints are listed (as row indices in the data matrix) together with the number of random walks which end in a particular endpoint. Pressing (4) moves the endpoints in current rectangular selection (*Selected terminal nodes* frame) to *Terminal nodes marked for further analysis*.

Once the user is satisfied with the endpoints selection, they can calculate their representation by pressing (5). The checkbox (6) switches between two options for the multiple

endpoints treatment (see subsection 4.3).

At this point the endpoints of interest are selected and the representation of all random walks ending in any of these endpoints are calculated. The user can switch to the pane *Homology classes by persistence selection* [1] and select homology classes they deem significant based on the persistence. On y axis of the diagram the *death/birth* ratio is plotted by default[4]. This ratio describes the significance of the hole corresponding to the homology class. The larger this value the sparser is the interior of the hole relative to the density of its boundary and the higher the significance for the random walk classification. On x axis the *death* value of the homology class is plotted - this value corresponds to the *size* of the corresponding hole and to the height of the branching on the dendrogram [4]/(13). For the artificial data used on the screenshots the situation is clear - there are two *essential classes* one corresponding to the larger hole in the data a second created by the selection of two disparate developmental fates[5].

The selection process is equivalent to the selection of endpoints and by pressing the button [2] the dendrogram [4] is updated.

On the dendrogram (13) the user can select leaves/branches to aggregate random walks into trajectories. Two trajectories can be built at the same time by selecting A/B trajectory (11). The selection process is the same as before - pressing "+" button (14) adds the walks in selection into the active trajectory, the "fire" button resets the trajectory. Note the slider (23) which adjusts the complexity of the dendrogram by thresholding on the minimal fraction of random walks for a node to be shown. The nodes below the threshold are merged.

The user can adjust the level of detail by the selection of significant homology classes

[4]as it gives better resolution for homology classes with large differences in size than more standard *death - birth*. This can be changed by the checkboxes [3].

[5]Note, that *essential classes* are infinite from the point of view of homology and their *death* values are chosen arbitrary for visualization purposes

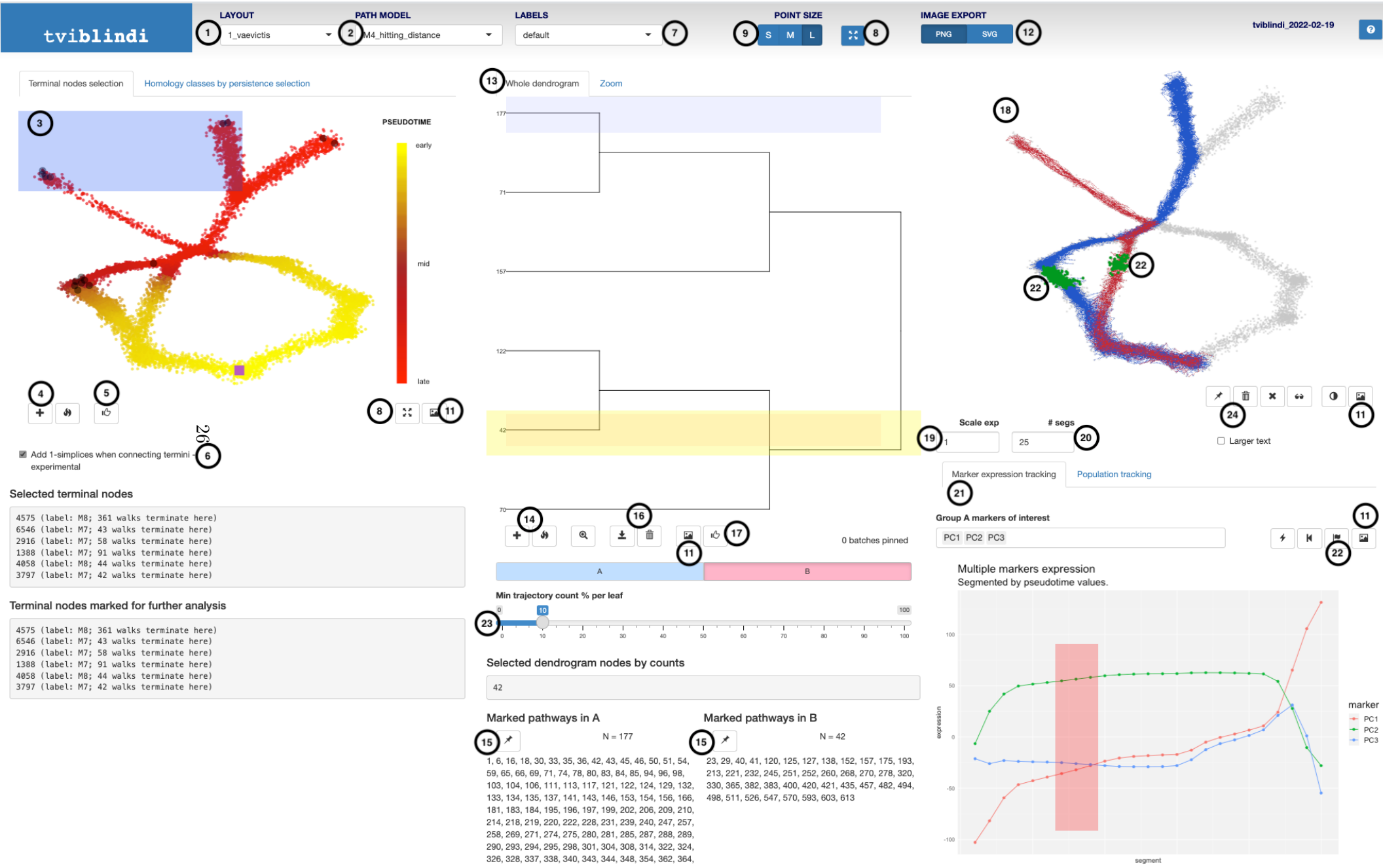
in [1]. In some cases it is more convenient to focus on a smaller subset of random walks. This can be achieved by pressing the button (17) - this makes the analysis focus only on random walks in the active trajectory and the user can analyze its topology in detail e.g. seeking for finer branching processes. To go back to the entire set of random walks, the user can press the "fire" button (4) and start over with endpoints selection.

Selected trajectories are visualized in 2D projection on the pane (18). The development of specific markers can be tracked at (21). The development of selected markers is visualized as lineplot connecting the average values of specific marker (or points on a specific random walks see below [5]) within a *pseudotime* segment. By default the *pseudotime* is divided into uniform segments (each containing same number of points) whose number can be set at (20). If the user wants to focus on either early or late stages of the development, they can adjust the scale at (19) the higher the value the more resolution is put on early stages and vice-versa. If the user is interested where in the 2D layout project certain part of development, they can select the segments and press (22). If multiple markers are tracked the points of all trajectories in selected segments are highlighted. If only one marker is tracked [5] we can see individual random walks and highlight only points from selected walks. If the user wants to manually clean selected trajectory based on the evolution of a specific marker they can select some random walks a remove them from the trajectory by pressing the "flash" button near [6] "flag".

It is often useful to export the trajectories/selected points for further analysis. The highlighted points can be pinned by pressing "pin" (24) (the "thrash bin" deletes all pinned points). The trajectories can be pinned (15). When pressing the "pin" the user is prompted to name the trajectory or points group. Pinned results are stored as list slots in the *tviblindi* object or can be exported as an fcs file by pressing the "download button" (16) (the "trash bin" button clears pinned trajectories). Once pressed the user is prompted for the name of

the exported fcs file and the results are stored either in a new fcs file built from the data matrix or added to existing fcs file (if provided - see section 6). The following is stored into the fcs file: active 2D layout, pseudotime from active "PATH MODEL", all sets of labels, all pinned trajectories and all pinned highlighted points.

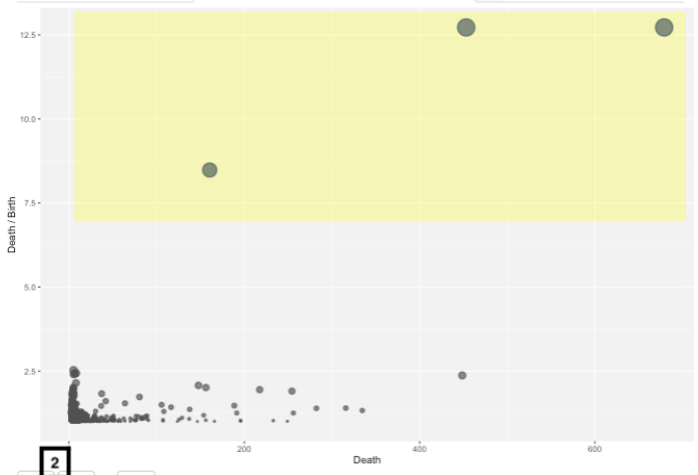
The button (11) saves the corresponding figure as either png or svg file (12).


tviblindi
LAYOUT
1_vaevictis
PATH MODEL
M4_hitting_distance
LABELS
default
POINT SIZE
S M L
IMAGE EXPORT
PNG SVG
tviblindi_2022-02-19
Terminal nodes selection
Homology classes by persistence selection
PSEUDOTIME
early
mid
late
Add 1-simplices when connecting termini – experimental
Selected terminal nodes
4575 (label: M8; 361 walks terminate here)
6546 (label: M7; 43 walks terminate here)
2916 (label: M7; 58 walks terminate here)
1388 (label: M7; 91 walks terminate here)
4058 (label: M8; 44 walks terminate here)
3797 (label: M7; 42 walks terminate here)
Terminal nodes marked for further analysis
4575 (label: M8; 361 walks terminate here)
6546 (label: M7; 43 walks terminate here)
2916 (label: M7; 58 walks terminate here)
1388 (label: M7; 91 walks terminate here)
4058 (label: M8; 44 walks terminate here)
3797 (label: M7; 42 walks terminate here)
Whole dendrogram
Zoom
177
71
157
122
42
70
0 batches pinned
A
B
Min trajectory count % per leaf
Selected dendrogram nodes by counts
42
Marked pathways in A
N = 177
1, 6, 16, 18, 30, 33, 35, 36, 42, 43, 45, 46, 50, 51, 54, 59, 65, 66, 69, 71, 74, 78, 80, 83, 84, 85, 94, 96, 98, 103, 104, 106, 111, 113, 117, 121, 122, 124, 129, 132, 133, 134, 135, 137, 141, 143, 146, 153, 154, 156, 166, 181, 183, 184, 195, 196, 197, 199, 202, 206, 209, 210, 214, 218, 219, 220, 222, 228, 231, 239, 240, 247, 257, 258, 269, 271, 274, 275, 280, 281, 285, 287, 288, 289, 290, 293, 294, 295, 298, 301, 304, 308, 314, 322, 324, 326, 328, 337, 338, 340, 343, 344, 348, 354, 362, 364,
Marked pathways in B
N = 42
23, 29, 40, 41, 120, 125, 127, 138, 152, 157, 175, 193, 213, 221, 232, 245, 251, 252, 260, 268, 270, 278, 320, 330, 365, 382, 383, 400, 420, 421, 435, 457, 482, 494, 498, 511, 526, 547, 570, 593, 603, 613
Scale exp
1
segs
25
Larger text
Marker expression tracking
Population tracking
Group A markers of interest
PC1 PC2 PC3
Multiple markers expression
Segmented by pseudotime values.
expression
segment
marker
PC1
PC2
PC3

tviblindi

LAYOUT: 1_vaevictis | PATH MODEL: M4_hitting_distance | LABELS: default | POINT SIZE: S M L | IMAGE EXPORT: PNG SVG | tviblindi_2022-02-19

Terminal nodes selection | Homology classes by persistence selection 1



2

Adjust for dissimilar densities

Death on x-axis 3

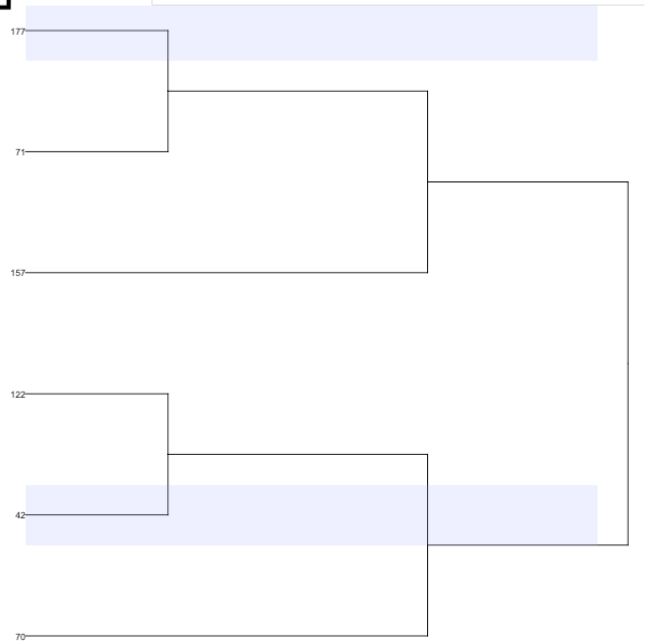
Selected homology classes

	Dimension	Birth	Death	BirthSimplex	DeathSimplex
2193	1	18.92563	160.5068	553289	1340778
2208	1	53.37344	678.9843	626	1528080
2209	1	35.61915	453.1251	1196175	1528081

Marked homology classes

	Dimension	Birth	Death	BirthSimplex	DeathSimplex
2193	1	18.92563	160.5068	553289	1340778
2208	1	53.37344	678.9843	626	1528080
2209	1	35.61915	453.1251	1196175	1528081

4 Whole dendrogram | Zoom



0 batches pinned

A | B

Min trajectory count % per leaf: 10

Selected dendrogram nodes by counts

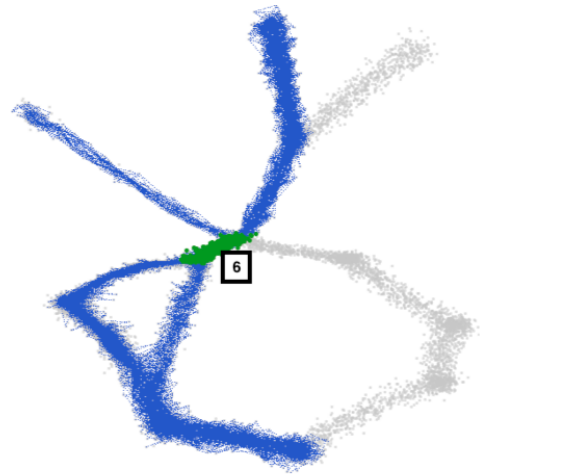
Marked pathways in A

N = 219

1, 6, 16, 18, 30, 33, 35, 36, 42, 43, 45, 46, 50, 51, 54, 59, 65, 66, 69, 71, 74, 78, 80, 83, 84, 85, 94, 96, 98, 103, 104, 106, 111, 113, 117, 121, 122, 124, 129, 132, 133, 134, 135, 137, 141, 143, 146, 153, 154, 156, 166, 181, 183, 184, 195, 196, 197, 199, 202, 206, 209, 210, 214, 218, 219, 220, 222, 228, 231, 239, 240, 247, 257, 258, 269, 271, 274, 275, 280, 281, 285, 287, 288, 289, 290, 293, 294, 295, 298, 301, 304, 308, 314, 322, 324, 326, 328, 337, 338, 340, 343, 344, 348, 354, 362, 364,

Marked pathways in B

N = 42



Scale exp: 1 | # segs: 25 | Larger text

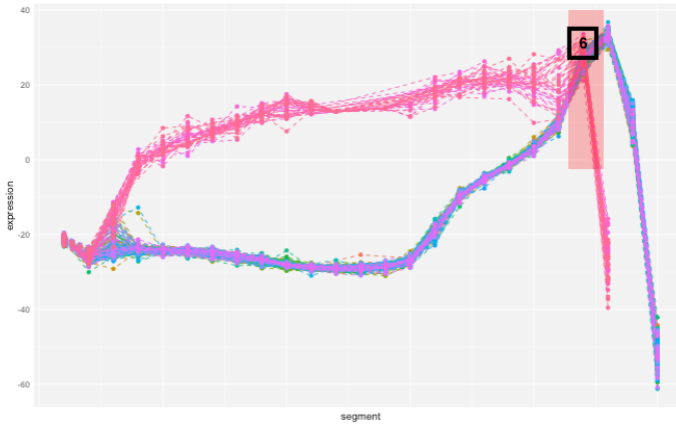
Marker expression tracking | Population tracking

Group A markers of interest: PC3 5

6

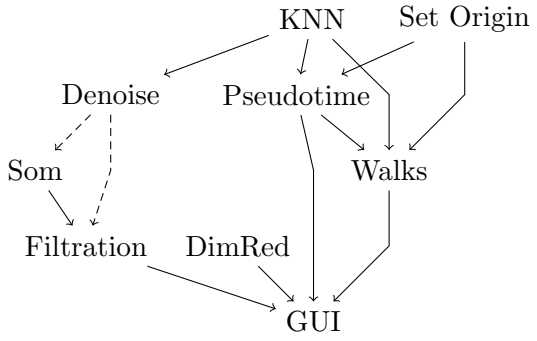
PC3 expression per walk: means per segment

Segmented by pseudotime values.



6 Running the analysis

The analytical pipeline consists of several steps which are not completely interdependent and can be (re)run in different order if necessary (see following graph of the dependence structure, dashed connection shows that the denoising step is optional)




```
tv1<-tviblindi(data=matrix,labels=character_vector,
               fcs=path_to_fcs,events_sel=selected_idices)

Set_origin(tv1,label=group_label,origin_name=path_model) # set the cell-of-
origin
KNN(tv1) #build KNN matrix
Denoise(tv1) #reduce noise before witness complex construction; for real world data
Som(tv1, xdim=15, ydim=15) #k-means clustering by default - 15*15 clusters
Filtration(tv1) # build witness complex
DimRed(tv1) # build low dimensional representation
Pseudotime(tv1,origin_name=path_model) # calculate pseudotime
Walks(tv1,N=5000,origin_name=path_model) #simulate 5000 random walks

launch_shiny(tv1) # launch GUI
```

All functions starting with capital letter are generic S3 methods and the documentation could be accessed via the class suffix e.g. `?KNN.tviblindi`. The parameter `origin_name` is optional and serves to distinguish different choices of cell-of-origin, *pseudotime* calculation and random walk simulation - see section 5.
The command

```
tv1<-tviblindi(data=matrix,labels=character_vector,
               fcs=path_to_fcs,events_sel=selected_idices)
```

deserves clarification. The `fcs`, `events_sel` parameters are optional and connect the *tviblindi* object to an existing fcs file to be enriched as a part of the analysis. If omitted the GUI will create a new fcs out of the data matrix when the export of a fcs file is requested. If these parameters are provided results of analysis will be added to an existing fcs file as additional columns. If only a subset of rows from the fcs file is used for the analysis, the parameter `events_sel` records the indices of these rows as an integer vector.

7 Performance evaluation

7.1 Comparison to state-of-the-art methods

Bibliography

Cannoodt, R., and W. Saelens, 2022: *dyntoy: Generating simple toy data of cellular differentiation*. R package version 0.9.9.

Carlsson, G., T. Ishkhanov, V. De Silva, and A. Zomorodian, 2008: On the local behavior of spaces of natural images. *International Journal of Computer Vision*, **76 (1)**, 1–12, doi:10.1007/s11263-007-0056-x.

Ding, J., A. Condon, and S. P. Shah, 2018: Interpretable dimensionality reduction of single cell transcriptome data with deep generative models. *Nature communications*, **9 (1)**, 1–13.

Edelsbrunner, H., and J. L. Harer, 2022: *Computational topology: an introduction*. American Mathematical Society.

Förster, Y.-P., L. Gamberi, E. Tzanis, P. Vivo, and A. Annibale, 2022: Exact and approximate mean first passage times on trees and other necklace structures: a local equilibrium approach. *Journal of Physics A: Mathematical and Theoretical*, **55 (11)**, 115 001, doi:10.1088/1751-8121/ac4ece, URL https://dx.doi.org/10.1088/1751-8121/ac4ece.

Hatcher, A., 2000: *Algebraic topology*. Cambridge Univ. Press, Cambridge, URL https://cds.cern.ch/record/478079.

Kachanovich, S., 2015: Witness complex. *GUDHI User and Reference Manual*, GUDHI Editorial Board, URL http://gudhi.gforge.inria.fr/doc/latest/group__witness__complex.html.

Pokorny, F. T., M. Hawasly, and S. Ramamoorthy, 2014: Multiscale Topological Trajectory Classification with Persistent Homology. *Robotics: Science and Systems*, doi:10.15607/rss.2014.x.054.

Rouvreau, V., 2015: Alpha complex. *GUDHI User and Reference Manual*, GUDHI Editorial Board, URL http://gudhi.gforge.inria.fr/doc/latest/group__alpha__complex.html.

Silva, V. d., and G. Carlsson, 2004: Topological estimation using witness complexes. *SPBG'04 Symposium on Point - Based Graphics 2004*, M. Gross, H. Pfister, M. Alexa, and S. Rusinkiewicz, Eds., The Eurographics Association, doi:10.2312/SPBG/SPBG04/157-166.

Szubert, B., J. E. Cole, C. Monaco, and I. Drozdov, 2019: Structure-preserving visualisation of high dimensional single-cell datasets. *Scientific reports*, **9 (1)**, 1–10.