This vignette is distributed with the R package **TraMineR**. Please cite as:
Gabadinho, A., G. Ritschard, N.S. Müller and M. Studer (2011). "Analyzing and Visualizing State Sequences in R with **TraMineR**." *Journal of Statistical Software*, **40**(4), 1–37. DOI 10.18637/jss.v040.i04

Analyzing and Visualizing State Sequences in R with TraMineR

Alexis Gabadinho University of Geneva
Gilbert Ritschard University of Geneva
Nicolas S. Müller University of Geneva
Matthias Studer University of Geneva

Abstract

This article describes the many capabilities offered by the **TraMineR** toolbox for categorical sequence data. It focuses more specifically on the analysis and rendering of state sequences. Addressed features include the description of sets of sequences by means of transversal aggregated views, the computation of longitudinal characteristics of individual sequences and the measure of pairwise dissimilarities. Special emphasis is put on the multiple ways of visualizing sequences. The core element of the package is the state sequence object in which we store the set of sequences together with attributes such as the alphabet, state labels and the color palette. The functions can then easily retrieve this information to ensure presentation homogeneity across all printed and graphical displays. The article also demonstrates how **TraMineR**'s outcomes give access to advanced analyses such as clustering and statistical modeling of sequence data.

Keywords: state sequences, categorical sequences, sequence visualization, sequence complexity, dissimilarities, optimal matching, representative sequences, R.

1. Introduction

This article is concerned with categorical sequence data and more specifically with state sequences, where the position of each successive state receives a meaningful interpretation in terms of age, date, or more generally of elapsed time or distance from the beginning of the sequence. Its aim is to examine a series of questions about such state sequences and to present the various solutions that we implemented in the R (R Development Core Team 2011) package **TraMineR** for answering them.

The addressed methods are for sets of sequences and most of them are holistic (Billari 2001b) in that they consider each sequence as a whole; i.e., as a conceptual unit. The discussion is mainly oriented towards the analysis of sequences describing individual life courses. Nevertheless, most of the discussed concepts and tools should be applicable in other domains such as text, biology, quality control or web logs analysis, to cite just a few.

Sequences are complex objects, and we need special tools for describing and displaying them. We consider, therefore, questions regarding the exploration and description of sets of sequences such as:

- Which characteristics of sequences are we interested in?
- What kind of indicators can we compute for a sequence set?
- What are suited plots for rendering sequences?
- How can we measure similarity between sequences?

With a more analytical or explanatory concern, we also consider issues such as:

- How can we identify groups with similar patterns and build typologies of sequences?
- How can we analyze the relationship of sequences with covariates?

In the social sciences, state sequences are of interest for studying life trajectories such as occupational histories, professional careers or cohabitational life courses. Some of the typical questions arising in this area are:

- Do life courses obey some social norm? Which are the standard trajectories? What kind of departures do we observe from these standards ? How do life course patterns evolve over time ?
- Why are some people more at risk to follow a chaotic trajectory or to stay stuck in a state? How does the trajectory complexity evolve across birth cohorts?
- How is the life trajectory related to sex, social origin and other cultural factors?

Empirical answers to such questions require us to consider collections of life sequences, to examine them from both a transversal and a longitudinal perspective, and to study their relationships with covariates.

The primary objective of sequence methods is then to extract simplified workable information from sequential data sets; that is, to efficiently summarize and render these sets and to categorize the sequential patterns into a limited number of groups. This is essentially an exploratory task that consists of computing summary indicators, as well as sorting, grouping and comparing sequences. The resulting groups and real-value indicators may then be submitted to classical inferential methods and serve, for instance, as response variables or explanatory factors for regression-like models.

A common approach for categorizing patterns consists of computing pairwise distances between them by means of sequence alignment algorithms (such as optimal matching) or other suitable metrics and using this information for clustering the sequences. This method has been applied to various data since the pioneering work of Abbott and Forrest (1986). A review can be found in Abbott and Tsay (2000). The expected outcome of such a strategy is a typology, with each cluster grouping cases with similar trajectories. Through binary logistic regression or classification trees, for example, we can then study how each cluster membership is related to covariates.

A more recent complementary approach considered in the literature (Elzinga and Liefbroer 2007; Widmer and Ritschard 2009) is to focus on sequence indicators measuring for instance the longitudinal diversity and complexity of the sequences and to analyze them by means of conventional statistical tools for real-value variables.

With a somewhat more aggregated point of view, an approach considered, for instance, by Billari (2001a) consists of looking at the sequence of transversal characteristics measured at each position, such as the diversity of states observed at each given age. Comparing the evolution of such transversal characteristics for different groups defined by birth cohorts or sex, for instance, provides instructive insights (Widmer and Ritschard 2009). However, when working with transversal indicators we lose the specific information on individual follow-ups.

We may indeed imagine many other ways of looking at categorical sequences such as correspondence analysis of the states (Deville and Saporta 1983) or advanced Markov modeling (Berchtold and Raftery 2002); i.e., the study of how the probability of a given state depends on the previously observed states. Transforming state sequences into event sequences and resorting to tools for mining frequent subsequences permits us to gain interesting knowledge about the typical sequencing of states or events (Billari, Frnkranz, and Prskawetz 2006; Ritschard, Gabadinho, Müller, and Studer 2008). A very common approach in the life course literature is *event history* or *survival* analysis (Mayer and Tuma 1990; Yamaguchi 1991; Hosmer and Lemeshow 1999; Blossfeld, Golsch, and Rohwer 2007) which focuses on the occurrence of a specific event or somewhat equivalently on the duration—time to event—until a given state transition. Though not addressed here, all these techniques may usefully complement the considered state sequence techniques.

The **TraMineR** R package is available from the Comprehensive R Archive Network at `http://CRAN.R-project.org/package=TraMineR` and offers many analysis and visualization tools for either state or event sequences. These tools include already known methods, as well as new developments. We focus in this article on the functions intended for state sequence analysis. The paper is organized as follows. In Section 2, we introduce the **TraMineR** library, describe the `mvad` data set used for illustration and give a first example of analysis that can be run with **TraMineR**. Section 3 defines the different forms of sequential data that are supported by the package. In Section 4, we introduce the central concept of *state sequence object*. Section 5 introduces two basic visualization tools and describes the general plotting principles used by the package. Section 6 is devoted to the summarization and visual rendering of sets of sequences, while Section 7 is concerned with individual sequence indicators. In Section 8, we present the metrics that were implemented for measuring pairwise dissimilarities between sequences. In Section 9, we illustrate how dissimilarities measures can be used for further statistical analysis. Finally, we make some concluding remarks in Section 10.

2. The TraMineR R package

TraMineR (Gabadinho, Ritschard, Studer, and Müller 2009) is a package for mining and visualizing sequences of categorical data describing life courses in R (R Development Core Team 2011), the name **TraMineR** being a contraction of Life Trajectory Miner for R. It puts together most of the features proposed separately by other software for sequential data and offers many original tools for managing, analyzing and rendering categorical sequences.

Other statistical programs that can handle state sequences include Abbott (1997)'s no-longer-maintained Optimize program, TDA (Rohwer and Ptter 2002), which is freely available at `http://www.stat.ruhr-uni-bochum.de/tda.html`, the add-on for Stata by Brzinsky-Fay, Kohler, and Luniak (2006), which is freely available for licensed Stata users, and the dedicated CHESA program by Elzinga (2007a), available from `http://home.fsw.vu.nl/ch.elzinga/`.

They all compute the optimal-matching edit distance between pairs of sequences and each of them offers specific useful facilities for describing sets of sequences. **TraMineR** is, to our knowledge, the first such toolbox for the free R statistical and graphical environment. Its salient characteristics are:

- R and **TraMineR** are free and open source;
- Since **TraMineR** is developed in R, it takes advantage of many already optimized procedures of R as well as of its powerful graphical capabilities;
- R runs under several OS including Linux, MacOS X, Unix and Windows; any R script with **TraMineR** functions runs unmodified under all operating systems;
- Specific **TraMineR** functions can be combined in the same script with any of the numerous basic statistical procedures of R as well as with those of any other R-package.

TraMineR is readily installed from within R via `install.packages("TraMineR")`. It features a unique set of procedures for analyzing and visualizing state sequence data, such as:

- Handling a large number of state sequence representations with simple functions for transforming to and from different formats;
- A whole series of easy to use plot functions for rendering sets of sequences (density plot, frequency plot, index plot, representative sequence plot and more);
- Individual longitudinal characteristics of sequences (length, time in each state, longitudinal entropy, complexity, turbulence and more);
- Sequence of transversal characteristics by position (transversal state distribution, transversal entropy, modal state);
- Other aggregated characteristics (transition rates, average duration in each state, sequence frequency);
- A choice of metrics for evaluating distances between sequences.

Table 1 gives an overview of the key functions for analyzing state sequences that will be described in the remainder of the article. It is worth mentioning that the package provides also tools for event sequences such as finding the most frequent and most discriminating subsequences and extracting association rules between subsequences, as well as tools for ANOVA-like analyses of sequences (Studer, Ritschard, Gabadinho, and Müller 2011) that will not be discussed here. See the User's guide (Gabadinho *et al.* 2009) for a detailed description of the package usage.

A first glance at ***TraMineR***

To illustrate how easy it is to work with **TraMineR** we consider the `mvad` example data set. This data frame is distributed with the library and will serve throughout the article. It contains the data used by McVicar and Anyadike-Danes (2002) for studying the school-to-work transition in Northern Ireland. The figures cover 712 individuals, the sequences being

Type	Function	Graphical function	Description
Data handling (Section 3)	`seqformat()` `seqconc()`, `seqdecomp()`		Translating between sequence formats Converting between character string and matrix representations of sequences

State sequence objects (Section 4)	`seqdef()`
	`alphabet()`
	`cpal()`
	`stlab()`
			Creating state sequence objects
			Setting and retrieving the alphabet
			Setting and retrieving the color palette
			Setting and retrieving the state labels

Individual sequences (Section 5)	`print()`
	
	`seqtab()`
		`seqiplot()`, `seqIplot()`
		`seqfplot()`
			Displaying and plotting individual sequences
			Sequence frequencies

Global and transversal descriptive statistics (Section 6)	`seqstatd()`
	
	`seqmeant()`
	`seqmodst()`
	`seqtrate()`
		`seqdplot()`, `seqHtplot()`
		`seqmtplot()`
		`seqmsplot()`
			Transversal state distributions and transversal entropies
			Mean durations in states of the alphabet
			Sequence of modal states
			Transitions rates

Sequence characteristics (Section 7)	`seqlength()`
	`seqdss()`
	`seqdur()`
	`seqsubsn()`
	`seqistatd()`
	`seqient()`
	`seqST()`
	`seqici()`
			Sequence lengths
			Distinct successive states by sequences
			Durations of the distinct successive states
			Number of subsequences by sequence
			Within sequence state distributions
			Within sequence entropies
			Within sequence turbulences
			Within sequence complexity indexes

Sequence dissimilarities (Section 8)	`seqsubm()`
	`seqdist()`
	
	`seqdistmc()`
	`seqrep()`
		
		`seqrplot()`
			Creating a substitution cost matrix
			Pairwise distances or distances to a baseline sequence
			Multichannel distances
			Extracting sets of non-redundant representative sequences

Table 1: **TraMineR**'s key functions.

their monthly follow-up over the course of 6 years starting in the month where they were first eligible to leave compulsory education (July 1993). Each individual is characterized by a unique identifier, 13 covariates and 72 monthly activity state variables from July 1993 to June 1999. Since the first two months of the follow-up are summer holidays, we look hereafter at trajectories from September 1993 yielding sequences of 70 monthly statuses. The states are school, FE (further education), employment, training, joblessness, and HE (higher education). See Table 2 for a description of the variables in `mvad`.

We first show how to build a typology of the observed school-to-work trajectories by clustering the sequences. This is done with the following few steps:

1. Load the library, retrieve the `mvad` data and create a state sequence object from the status variables (columns 17 to 86):

id	Unique individual identifier
weight	Sample weights
male	Binary dummy for gender; 1 = male
catholic	Binary dummy for community; 1 = Catholic
Belfast	Binary dummies for location of school, one of five Education and Library Board areas
N.Eastern	see Belfast
Southern	see Belfast
S.Eastern	see Belfast
Western	see Belfast
Grammar	Binary dummy indicating type of secondary education; 1 = grammar school
funemp	Binary dummy indicating father's employment status at time of survey; 1 = father unemployed
gcse5eq	Binary dummy indicating qualifications gained by the end of compulsory education; 1 = 5 or more GCSEs at grades A-C, or equivalent
fmpr	Binary dummy indicating SOC code of father's current or most recent job; 1 = SOC1 (professional, managerial or related)
livboth	Binary dummy indicating living arrangements at time of first sweep of survey (June 1995); 1 = living with both parents
jul93	Monthly activity variables coded 1-6; 1 = school, 2 = FE, 3 = employment, 4 = training, 5 = joblessness, 6=HE
⋮	⋮
jun99	see jul93

Table 2: List of variables in the `mvad` data set.

```
R> library("TraMineR")
R> data("mvad")
R> mvad.alphab <- c("employment", "FE", "HE", "joblessness",
+       "school", "training")
R> mvad.seq <- seqdef(mvad, 17:86, xtstep = 6, alphabet = mvad.alphab)
```

2. Compute pairwise optimal matching (OM) distances between sequences with an insertion/deletion cost of 1 and a substitution cost matrix based on observed transition rates:

```
R> mvad.om <- seqdist(mvad.seq, method = "OM", indel = 1, sm = "TRATE")
```

3. Proceed to an agglomerative hierarchical clustering using the obtained distance matrix, select the four-clusters solution and express it as a factor:

```
R> library("cluster")
R> clusterward <- agnes(mvad.om, diss = TRUE, method = "ward")
R> mvad.cl4 <- cutree(clusterward, k = 4)
R> cl4.lab <- factor(mvad.cl4, labels = paste("Cluster", 1:4))
```

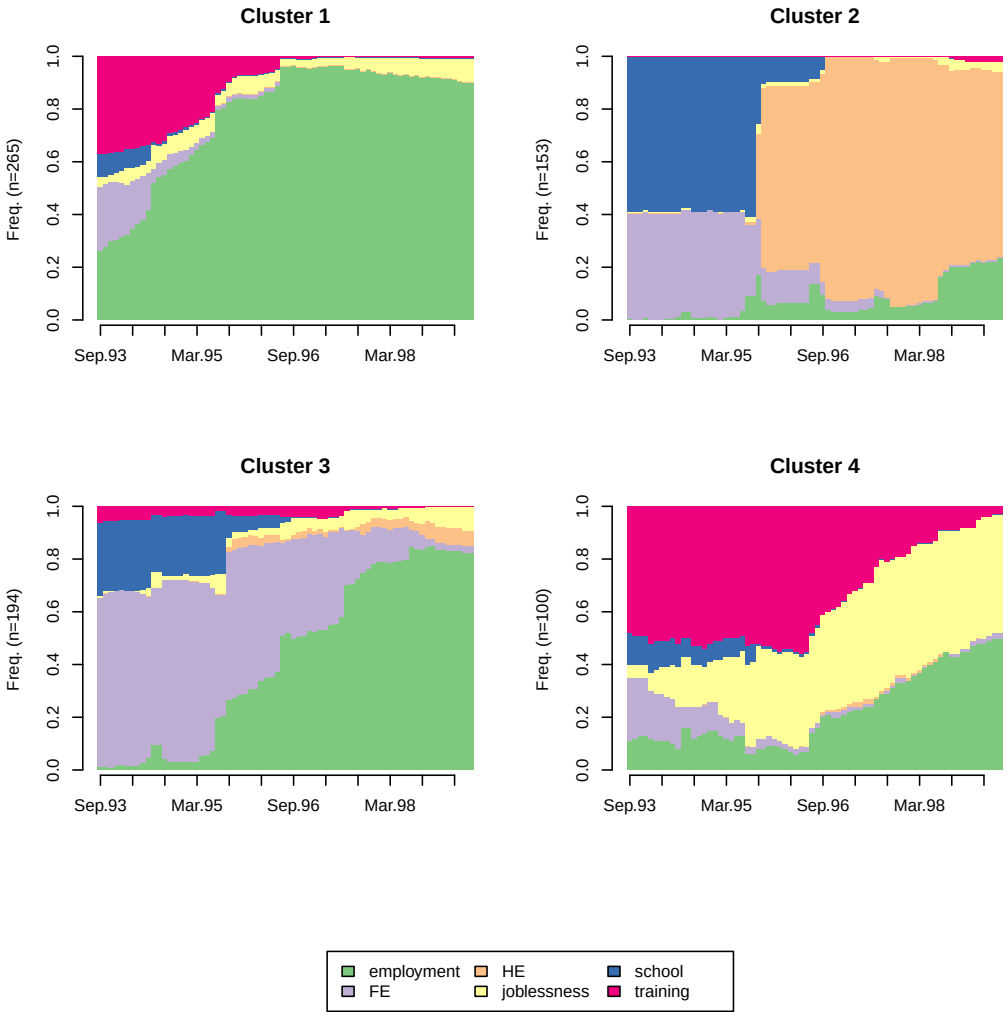

Figure 1: State distribution plots by cluster.

4. Visualize the cluster patterns by plotting their transversal state distributions:

```
R> seqdplot(mvad.seq, group = cl4.lab, border = NA)
```

We see in Figure 1 that the first cluster is formed essentially by the youngsters who join the workforce soon after ending compulsory school, while those who pursue education are in the second (higher education) and third clusters. The last cluster groups less successful transitions from school to work with long spells of joblessness or training.

To continue, we examine how the diversity of states within each sequence is related to sex, to whether the father is unemployed and to whether the qualification grade at end of compulsory school was good. We compute the longitudinal entropy and regress it on the covariates:

```
R> entropies <- seqient(mvad.seq)
R> lm.ent <- lm(entropies ~ male + funemp + gcse5eq, mvad)
```

	Estimate	Std error	t value	Pr($>|t|$)
(Intercept)	0.39	0.01	32.73	0.00
Male	-0.04	0.01	-3.03	0.00
Father unemployed	0.03	0.02	1.67	0.10
Good end compulsory school grade	0.07	0.01	4.90	0.00

Table 3: Regressing entropies on a selection of covariates.

Results show that males experience less diverse states and that youngsters with good grades at the end of compulsory school experience more diverse states. Whether the father is unemployed does not have a significant effect.

3. Sequence representations

State sequences can be represented in many different ways, depending on the data source and on how the information is organized. Data organization and conversion between formats is discussed in detail in Ritschard, Gabadinho, Studer, and Müller (2009), where an ontology of longitudinal data presentations is given that may help identify the kind of data at hand. Here, we limit the discussion to the sequence data representations that **TraMineR** can handle and import. Those formats are listed in Table 4 together with the conversions that can currently be done with the provided `seqformat()` function.

3.1. State sequences

We consider sequences of discrete or categorical data. Formally, we define a state sequence of length ℓ as an ordered list of ℓ elements successively chosen from a finite set A of size $a = |A|$ that is called the *alphabet*. A natural way of representing a sequence x is by listing the successive elements that form the sequence $x = (x_1, x_2, \ldots, x_\ell)$, with $x_j \in A$. With reference to this expanded form of sequences, *state sequences* are characterized by two properties. Firstly, they are formed by elements that are states; i.e., something that can last as opposed, for instance, to events that occur at given time points. Secondly, the position of each element conveys meaningful information in terms of age, date or, more generally, elapsed time or distance from the beginning of the sequence. Position indexes providing time information may be either absolute *calendar* values (day, year, month, ...) or relative *process time* (age, process duration, ...).

In **TraMineR**, the expanded form is called *STate-Sequence* (STS) format. In this format, the successive states (statuses) of an individual are given either in consecutive columns, or as a character string with states separated by a given symbol such as '-' or '/', the former being the default separator. Each position (column) is supposed to correspond to a predetermined time unit.

3.2. Other sequence representations

Sequence data can be represented in more compact ways than STS essentially by giving only one of several same successive states. In that case, we have to explicitly stamp the successive distinct states with their starting position or duration. Table 4 displays the same example

Code	Conversion	Example
STS	from/to	*Id* 18 19 20 21 22 23 24 25 26 27 101 S S S M M MC MC MC MC D 102 S S S MC MC MC MC MC MC MC
SPS	from/to	*Id* 1 2 3 4 101 (S,3) (M,2) (MC,4) (D,1) 102 (S,3) (MC,7)
DSS	to	*Id* 1 2 3 4 101 S M MC D 102 S MC
SPELL	from	*Id* *Index* *From* *To* *State* 101 1 18 20 S (single) 101 2 21 22 M (married) 101 3 23 26 MC (married with children) 101 4 27 27 D (divorced) 102 1 18 20 S (single) 102 2 21 27 MC (married with children)

Table 4: Sequence data representations; some formats handled by the `seqformat()` function.

of two sequences in the different formats. The two considered sequences describe family formation histories of two individuals, the states being single (S), married (M), married with children (MC) and divorced (D).

A first efficient way of representing a state sequence is by listing the distinct successive states with their associated durations. We get thus a sequence of couples (x_j, t_j) where x_j is a state and t_j its duration; i.e., the number of times it is repeated. This is the *State-Permanence-Sequence* (SPS) format (Aassve, Billari, and Piccarreta 2007). By considering only the distinct successive states without their associated durations, we get the *Distinct-Successive-States* (DSS) sequence format. This DSS form holds the basic state sequencing information, but loses all time (t_j) and, more generally, alignment data.

In the SPELL format there is one line for each spell. Each spell is characterized by the state (supposed constant during the spell) and the spell start and end times. STS sequences can easily be derived from such representation.

4. State sequence objects

The general philosophy of the library is to ensure that the various results and plots outputted for a same set of sequence data use the same state labels and colors. Likewise, any information on case weights and possible missing information about some positions in sequences should be treated the same way throughout the analysis. To achieve this goal, the **TraMineR** functions for state sequence analysis require as an argument a *state sequence object* that includes both the sequential data and its attributes. Thus, the first step when using **TraMineR** for state sequence analysis is to create a *state sequence object*. This is done with the `seqdef()` function from data organized in either of the STS, SPS or SPELL forms described in the previous section. We show below how to create a state sequence object from the `mvad` data

Attribute	Description	Argument	Default	Retrieve/set
alphabet	List of states	`states`	From input data	`alphabet()`
cpal	Color palette	`cpal`	**RColorBrewer** palette	`cpal()`
labels	Long state labels	`labels`	From input data	`stlab()`
cnames	Position names	`cnames`	From input data	`names()`
row.names	Row (sequence) labels	`id`	From input data	`rownames()`
weights	Optional case weights	`weights`	`NULL`	

Table 5: Main sequence object attributes.

set introduced in Section 2. The main attributes are listed in Table 5, together with their default values and the dedicated functions to retrieve or set them.

4.1. Creating state sequence objects

In the `mvad` data set, the retained activity status variables are stored in columns 17 to 86. We display these statuses for the first six considered months (September 1993 to February 1994) of the first two records:

```
R> mvad[1:2, 17:22]
```

```
      Sep.93     Oct.93     Nov.93     Dec.93   Jan.94   Feb.94
1 employment employment employment employment training training
2         FE         FE         FE         FE       FE       FE
```

The default input format for the `seqdef()` function is STS, which is appropriate for the `mvad` data set. If the input data is in another format it must be specified with the `informat` argument and `seqdef()` will automatically make the required conversion.

Alphabet and state labels

The alphabet is the list of states allowed in the sequences. Both short and long labels of the states forming the alphabet are attached to the object. Long labels serve mainly for color legends in plots, while short state names are primarily used in printed outputs. Shorter names produce cleaner and shorter output when printing the sequences.

By default, the sorted list of the distinct states found in the input data (as returned by the `seqstatl()` function) defines the alphabet and is used as state names and labels. This can be changed with optional arguments, which is necessary, for example, when the alphabet contains states that do not appear in the retained sequences.

Below we specify short state names with the `states` argument and long state labels with `labels`. These arguments expect vectors of names or labels that are ordered conformably with the alphabet. The `alphabet` argument can be used to change the order of the states, in which case the vectors passed with the `states` and `labels` arguments should conform to the newly defined order.

```
R> mvad.lab <- c("Employment", "Further education", "Higher education",
+       "Joblessness", "School", "Training")
R> mvad.scode <- c("EM", "FE", "HE", "JL", "SC", "TR")
R> mvad.seq <- seqdef(mvad, 17:86, alphabet = mvad.alphab, states = mvad.scode,
+       labels = mvad.lab, xtstep = 6)
```

Now the sequences are stored in the `mvad.seq` sequence object. We can display them in the concise SPS representation with:

```
R> print(mvad.seq[1:5, ], format = "SPS")
```

```
  Sequence
1 (EM,4)-(TR,2)-(EM,64)
2 (FE,36)-(HE,34)
3 (TR,24)-(FE,34)-(EM,10)-(JL,2)
4 (TR,47)-(EM,14)-(JL,9)
5 (FE,25)-(HE,45)
```

4.2. Other important attributes and properties

We briefly comment here upon some other important attributes that will be used in conjunction with the alphabet and state labels by **TraMineR**'s functions.

State colors and position names

The sequence plot functions provided by the library need a distinct color for each state. A *color palette* is therefore attached to the sequence object. A default color palette from **RColorBrewer** (Neuwirth 2007) is automatically selected as long as the alphabet size does not exceed 12. *Position names* serving mainly for labeling the ticks of the x-axis but that are also useful for increasing readability of tabulated output are also an attribute of the object. If left unspecified, position names are taken from the corresponding column names of the original data frame. The interval between the x-axis tick-marks is an additional attribute that can be set (`xtstep` argument) for optimizing rendering.

Case weights

Survey data often come with *case weights* that account for the sampling scheme and unit non-responses. Using such case weights is important to compensate for sampling bias and thus get results that are more realistic. When weights are attached to the state sequence object, the **TraMineR** functions that can handle weights automatically produce weighted results. To disable the use of weights, add option `weighted=FALSE` to the function.

The `weight` variable in `mvad` contains case weights that account for the selective attrition during the survey and we attach them to the sequence object as shown below. Unless otherwise specified, we will use this weighted sequence object from here on.

```
R> mvad.seq <- seqdef(mvad, 17:86, alphabet = mvad.alphab, states = mvad.scode,
+       labels = mvad.lab, weights = mvad$weight, xtstep = 6)
```

Missing values

Missing values in the expanded (STS) form of a sequence occur, for example, when:

- Sequences do not start on the same date while using a calendar time axis;
- The follow-up time is shorter for some individuals than for others yielding sequences that do not end up at the same position;
- The observation at some positions is missing due to nonresponse, yielding internal gaps in the sequences.

The way missing values should be handled may be different for each of these situations. In the first case, we may want to maintain explicitly the starting missing values to preserve alignment across sequences or possibly left-align sequences by switching to a process time axis. In the second case, the ending missing terms could just be ignored.

To allow such differentiated treatments, **TraMineR** distinguishes *left*, *in-between* and *right* missing values. We can specify how each of the missing types should be encoded with the `left`, `gaps` and `right` arguments. By default, gaps and left-missing states are coded as explicit missing values while all missing values encountered after the last valid (rightmost) state in a sequence are considered void elements; i.e., the sequence is considered to end after the last valid state.

The specific treatment of each type of missing value will depend upon whether the analysis method envisaged supports missing values; and, if yes, which kind it supports. Most of the proposed functions, such as `seqdist()` for computing distances between sequences, have optional arguments for dealing with missing states.

Subsets and attributes inheritance

Subsets of sequence objects can be defined by specifying row and column indexes (or names) as for R matrices and data frames. Every subset of a state sequence object inherits its 'parent' attributes. The alphabet and color palette, for instance, remain the same for all subsets. This is of particular importance when comparing graphics that render different subsets of a same sequence object.

5. Visualizing individual state sequences

State sequence visualization is one of the most important features of the package. This section introduces two basic plotting functions, namely the index plot intended to render a set or subset of individual state sequences and the frequency plot that visualizes them according to their frequencies. We explain also the common design of most of **TraMineR**'s plotting functions.

5.1. Sequence index plots

A *sequence index plot* (Figure 2) individually renders the selected state sequences. Each of them is represented by horizontally stacked boxes that are colored according to the state at the successive positions. The resulting bars are put above each other to vertically align the

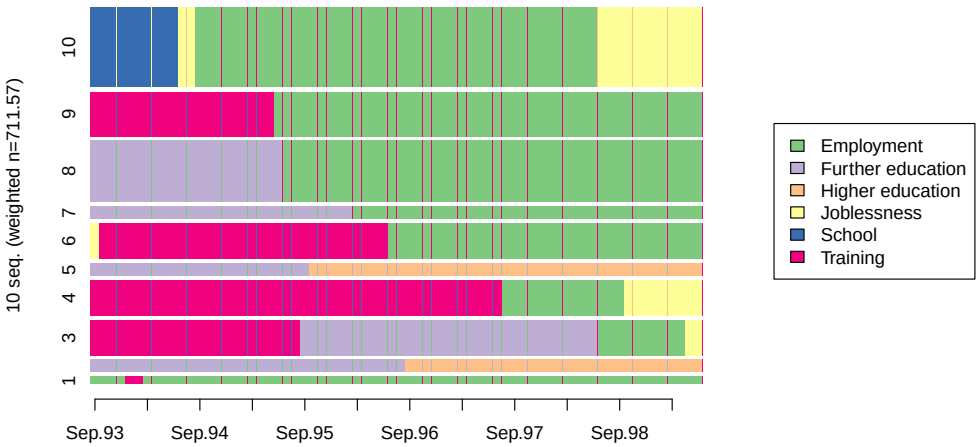

Figure 2: Sequence index plot of sequences 1 to 10.

positions. We thus visualize, for each case, the individual longitudinal succession of states as well as, through the length of each color segment, the duration spent in each successive state. The alignment also permits easy transversal comparisons at each position. The *sequence index plot* shown in Figure 2 was obtained with the command below:[1]

```
R> seqiplot(mvad.seq, border = NA, with.legend = "right")
```

Since we have attached case weights to the `mvad.seq` sequence object, the width of the bar representing each sequence is proportional to its weight. This default behavior could be changed with the `weighted=FALSE` argument. The plotted sequences are selected with the `idxs` argument by providing either a vector of indexes, or 0 for requesting all the sequences. The default value is `1:10` and Figure 2 displays therefore only the first 10 sequences of `mvad.seq`.

The `seqIplot()` alias produces full index plots that display all the sequences in the set without spaces between sequences and without borders around unit states. The usefulness of such plots has, for instance, been stressed by Scherer (2001) and Brzinsky-Fay *et al.* (2006). However, when the number of displayed sequences is large, they may produce burden pictures that are often hard to interpret.[2] We can partially overcome this drawback by sorting the sequences according to the values of a suitably chosen covariate—passed with the `sortv` argument. Good choices are, for instance, the distance to the most frequent sequence or the scores of a multidimensional scaling analysis[3] of the dissimilarities between sequences

[1] `seqiplot()`, as most other plotting functions described in this paper, is just an alias for calling a generic `seqplot()` state sequence plot function with the appropriate `type` argument and suitable default option values. The `border=NA` option suppresses the border that surrounds, by default, each unit state in the sequence.

[2] When plotting several hundred of sequences, saving index plots may also produce heavy files in vectorial formats such as PostScript and PDF; generating plots in bitmap formats such as PNG or JPEG is recommended in such cases.

[3] The scores are obtained from the dissimilarity matrix with the `cmdscale()` function.

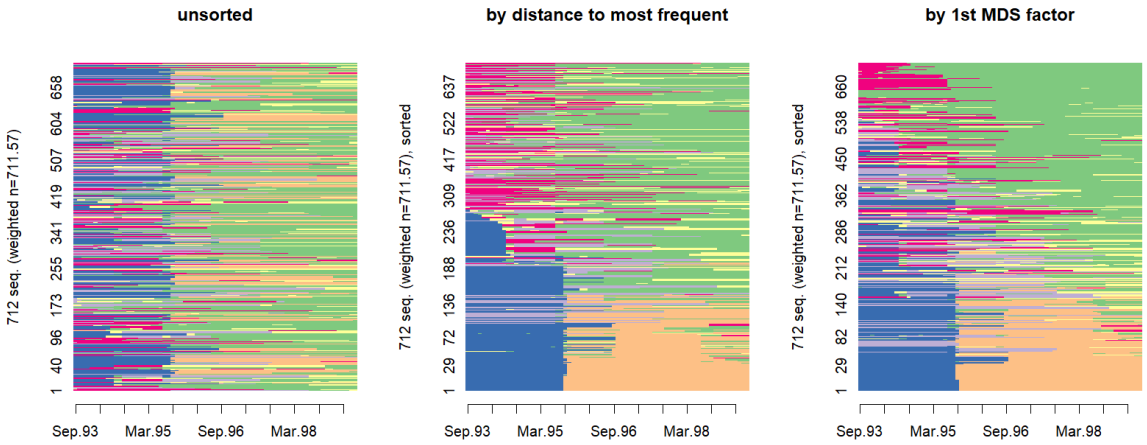

Figure 3: Unsorted and sorted full-sequence index plots.

(Figure 3). Both solutions suppose that we can compute such dissimilarities; this will be addressed in Section 8.

5.2. Sequence frequencies

The `seqtab()` function returns a table with the counts and percent frequencies of the sequences sorted in decreasing order of their frequencies. In the next example, we request the four most frequent sequences of `mvad.seq` with `idxs=1:4`. In the printed outcome, sequences are displayed in the shorter and more readable SPS format:

```
R> seqtab(mvad.seq, idxs = 1:4)
```

```
             Freq Percent
SC/24-HE/46    33     4.7
SC/25-HE/45    25     3.5
TR/22-EM/48    18     2.5
EM/70          15     2.1
```

The most frequent sequence in the `mvad.seq` object is a spell of two years of school followed by 46 months of higher education. It accounts, however, for only 4.7% of the total weights of the 712 cases considered. The second most frequent sequence, which concerns 3.5% of the weighted individuals, is indeed very similar to the previous one.

Sequence frequency plots

A graphical view of the sequence frequency table where bar widths are proportional to the frequencies is obtained with the `seqfplot()` function. Figure 4 shows the plot of the weighted and unweighted frequencies[4] obtained with:

[4]Remember that we attached case weights to our sequences; the frequencies are thus weighted by default. Since we have to issue two separate `seqfplot()` commands, we use `par(mfrow=...)` for arranging the plots and, therefore, we deactivate the automatic display of the legend that is not compatible with it (see below).

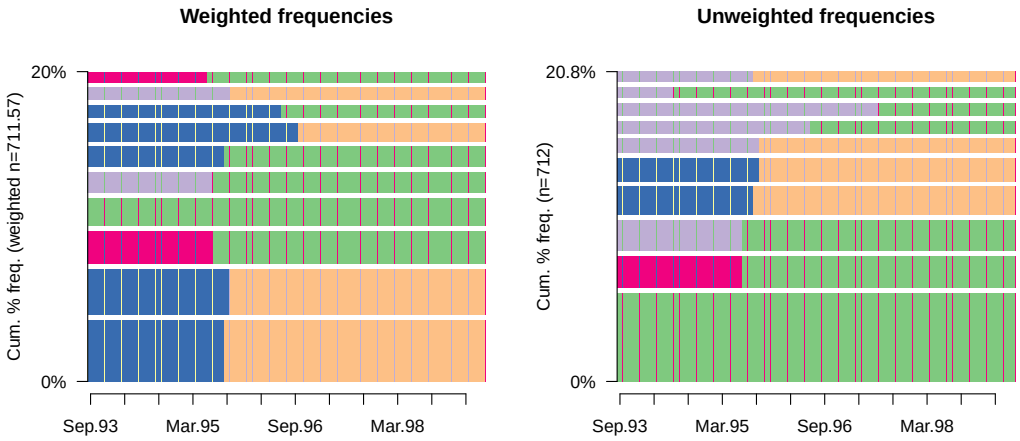

Figure 4: Weighted and unweighted sequence frequency plots.

```
R> par(mfrow = c(1, 2))
R> seqfplot(mvad.seq, border = NA, with.legend = FALSE,
+       main = "Weighted frequencies")
R> seqfplot(mvad.seq, weighted = FALSE, border = NA,
+       with.legend = FALSE, main = "Unweighted frequencies")
```

By default, only the 10 most frequent sequences are shown. The y-axis indicates the cumulative percentage of the represented sequences. If we look at the unweighted results, the most frequent sequence is to stay employed during the entire follow-up period (be in state EM during 70 months). This sequence, which was the fourth most frequent in the weighted frequency table with 2.5% of the total weight, accounts for 7% of the 712 cases considered.

The probability for two individuals to follow exactly the same 70-month trajectory is small, yielding a large number of different patterns. The 10 most frequent sequences account for only about 20% of all the trajectories, which reflects this high diversity.

5.3. Reading and controlling state sequence plots

The way index plots render individual state sequences with horizontally stacked boxes is common to other functions of the library that visualize specific state sequences. The position in the sequence is read on the x-axis. The first value on this axis is the selected origin. The sequence is read from left to right in the same way as printed outputs. Tick labels for the x-axis are retrieved, by default, from the plotted sequence object.

The values on the y-axis are the indexes of the plotted sequences. The index refers to the considered ranking of the sequences. For instance, in *sequence index plots*, the default order is that in the state sequence object unless a specific sort variable is provided with the `sortv` argument. In *sequence frequency plots*, sequences are sorted according to their frequency in the data set, while in *representative sequence plots* (Section 9.1), sequences are sorted according to their representativeness score.

The indexes on the y-axis—and hence the sequences—are displayed bottom-up. Thus, when sequences are sorted, the first ranked one is at the bottom of the plot.[5] This respects the usual standard for y-axes. It may, however, be confusing when compared with the corresponding printed outputs where sequences are displayed top-down.

Other aspects of the graphic (title, font size, axes display, axis label, state legend, ...) can be controlled with dedicated options described in detail in the reference manual. There is also an option to produce separate plots by levels of a covariate.

6. Computing and plotting overall and transversal statistics

We now turn to the facilities offered by **TraMineR** for visualizing and computing overall and transversal descriptive statistics of a set of sequences. The functions discussed here all require a state sequence object as main argument and admit a series of optional parameters. We illustrate with the `mvad.seq` weighted sequence object created on page 11.

6.1. Overall statistical characteristics

We consider, first, global synthesized information that is based neither on individual longitudinal characteristics nor on transversal characteristics by position. More specifically, we focus on the overall state distribution and transition rates between states.

Mean time spent in each state

A first synthetic information is given by the mean—not necessarily consecutive—time spent in the different states; that is, the mean number of times each state is observed in a sequence. This characterizes the overall state distribution. As an example, we plot the mean times for two subsets defined by the `funemp` covariate that indicates whether the respondent's father was unemployed at the time of the survey (Figure 5). The graphic with the distinct plots by levels of the `funemp` covariate is obtained by passing `funemp` as `group` argument to the plotting function. This option is common to all the plotting functions presented in this article.

```
R> seqmtplot(mvad.seq, group = mvad$funemp, ylim = c(0, 30))
```

We can see that the mean time spent in joblessness and training is higher for interviewees with unemployed fathers, while the time they spent in 'school', 'further education' and 'higher education', is lower.

Mean time values are obtained with the `seqmeant()` function.[6] However, unlike for graphical displays, functions returning statistics and sequence characteristics do not have a `group` argument. We can retrieve the values by levels of a covariate with the row indexing mechanism[7] or with the `by()` function:

```
R> by(mvad.seq, mvad$funemp, seqmeant)
```

[5] This can be changed by with the `idxs` argument.
[6] Individual time spent in each state can be obtained with `seqistatd()`.
[7] `seqmeant(mvad.seq[mvad$funemp=="yes",])` and `seqmeant(mvad.seq[mvad$funemp=="no",])`.

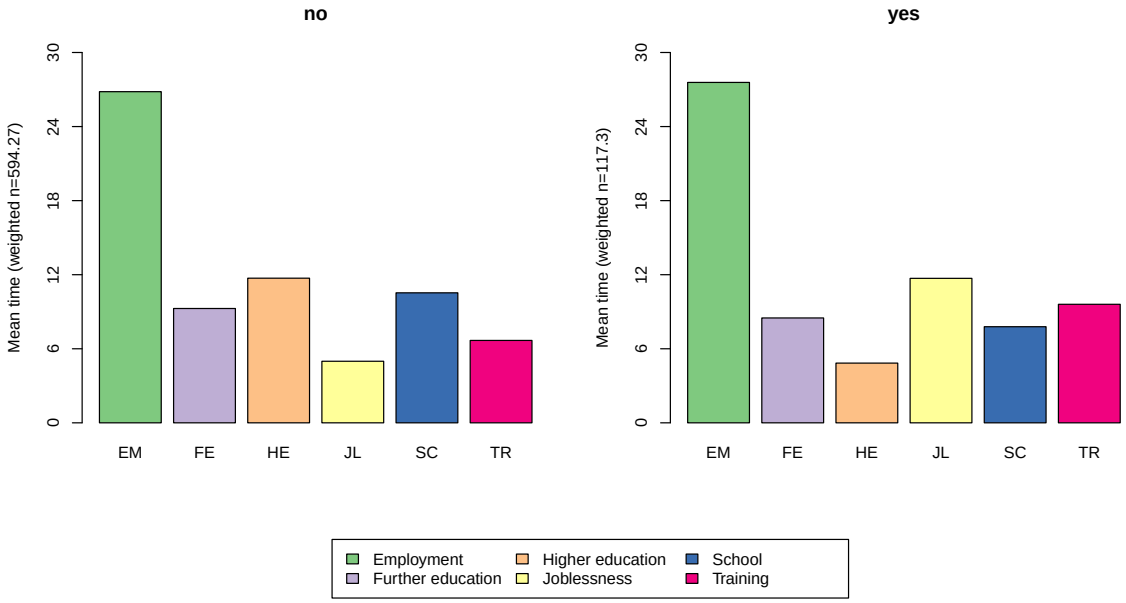

Figure 5: Mean time spent in each state by father's unemployment status.

Transition rates

Another interesting information about a set of sequences is the transition rate between each couple of states (s_i, s_j); i.e., the probability to switch at a given position from state s_i to state s_j. Let $n_t(s_i)$ be the number of sequences that do not end in t with state s_i at position t and let $n_{t,t+1}(s_i, s_j)$ be the number of sequences with state s_i at position t and state s_j at position $t+1$. The transition rate $p(s_j \mid s_i)$ between states s_i and s_j is obtained as

$$p(s_j \mid s_i) = \frac{\sum_{t=1}^{L-1} n_{t,t+1}(s_i, s_j)}{\sum_{t=1}^{L-1} n_t(s_i)} \quad .$$

with L the maximal observed sequence length.

The `seqtrate()` function returns the matrix of transition rates for the provided sequence object. By default, the rates are assumed position-independent; i.e., the same whatever t. The outcome is a single matrix where each row i gives a transition distribution from the originating state s_i in t to the states in $t+1$; that is, each row total equals one. Hence, transition rates provide information about the most frequent state changes observed in the data together with, on the diagonal, an assessment of the stability of each state.

In the following example we compute the transition rate matrix for the `mvad.seq` sequence object:

```
R> mvad.trate <- seqtrate(mvad.seq)
R> round(mvad.trate, 2)
```

```
        [-> EM] [-> FE] [-> HE] [-> JL] [-> SC] [-> TR]
[EM ->]    0.99    0.00    0.00    0.01    0.00    0.00
[FE ->]    0.03    0.95    0.01    0.01    0.00    0.00
[HE ->]    0.01    0.00    0.99    0.00    0.00    0.00
[JL ->]    0.04    0.01    0.00    0.94    0.00    0.01
[SC ->]    0.01    0.01    0.02    0.01    0.95    0.00
[TR ->]    0.04    0.00    0.00    0.01    0.00    0.94
```

We learn from this outcome that the highest transition rates (0.04) are observed between states JL (joblessness) and EM (employment), and between states TR (training) and EM. Moreover, states JL and TR are the most unstable with a probability of 0.06 ($1.0 - 0.94$) to leave the state at each position t.

Time-varying transition rates can be obtained with option `time.varying=TRUE`, in which case a 3-dimensional array with a distinct transition rate matrix for each of the positions $t = 1, 2, \ldots, L-1$ is returned. The matrix for position t is computed by considering only the states at t and $t+1$. The third dimension of the array corresponds to the position t index.

6.2. Transversal state distributions

Time varying transition rates are transversal characteristics computed at the successive considered positions. In the same vein, it is of interest to look at the transversal distribution of the states at each position of the considered sequences. The `seqstatd()` function returns a $a \times L$ table containing in each column the distribution among the a states of the alphabet at the corresponding position in the sequence. The output of this function for the first height months in `mvad.seq` is shown below:

```
R> seqstatd(mvad.seq[, 1:8])
```

```
    [State frequencies]
   Sep.93 Oct.93 Nov.93 Dec.93 Jan.94 Feb.94 Mar.94 Apr.94
EM  0.034  0.036  0.044  0.051  0.055  0.058  0.066  0.067
FE  0.223  0.217  0.215  0.215  0.209  0.206  0.204  0.200
HE  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
JL  0.048  0.055  0.051  0.050  0.058  0.068  0.064  0.067
SC  0.472  0.466  0.464  0.461  0.451  0.451  0.446  0.446
TR  0.223  0.227  0.226  0.223  0.227  0.218  0.221  0.220

     [Valid states]
   Sep.93 Oct.93 Nov.93 Dec.93 Jan.94 Feb.94 Mar.94 Apr.94
N     712    712    712    712    712    712    712    712

     [Entropy index]
   Sep.93 Oct.93 Nov.93 Dec.93 Jan.94 Feb.94 Mar.94 Apr.94
H    0.72   0.73   0.73   0.74   0.75   0.76   0.77   0.77
```

State distribution plot

The `seqdplot()` function generates a graphical view of the (weighted) state distributions. We illustrate by generating plots by values of the *gcse5eq* (qualification gained at end of compulsory school) covariate:

```
R> seqdplot(mvad.seq, group = mvad$gcse5eq, border = NA)
```

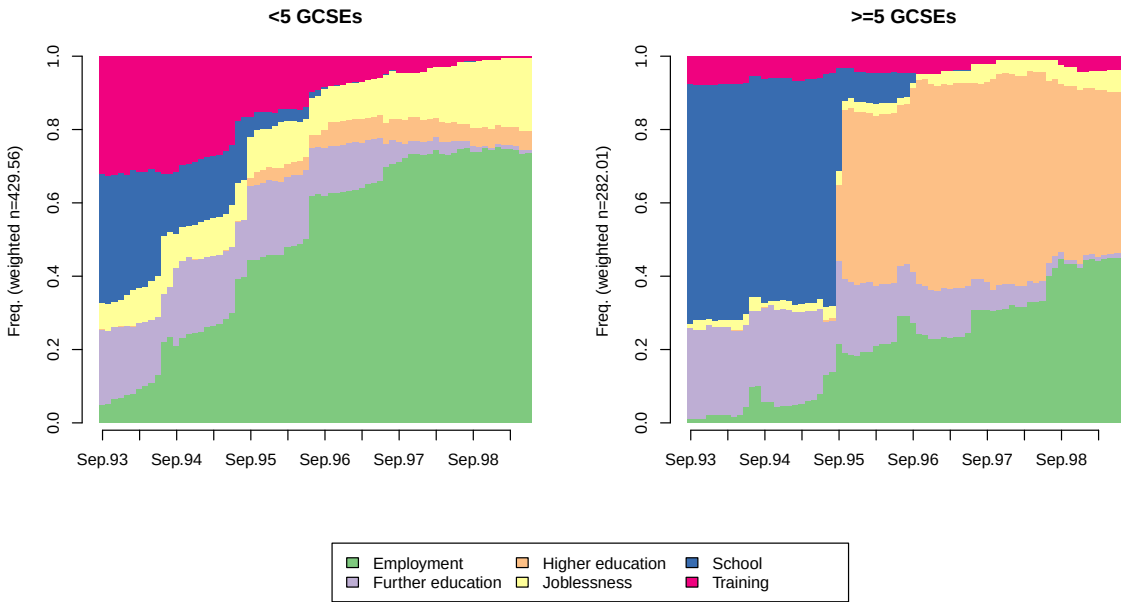

Figure 6: Transversal state distributions by end of compulsory school qualification group.

The result shown in Figure 6 exhibits significant jumps in the sequence of state distributions. As commented by McVicar and Anyadike-Danes (2002), they correspond mainly to the beginning and ending of education cycles.

A state distribution plot, as produced by `seqdplot()`, displays the general pattern of the whole set of trajectories. When interpreting such graphics, one must remember that, unlike sequence index plots and sequence frequency plots, they do not render individual sequences or individual follow-ups. They provide aggregated views made of successive slices, each of which represents transversal characteristics.

Sequence of modal states

An interesting summary that can be derived from the state distributions is the sequence made of the most frequent state at each position. It is obtained with the `seqmodst()` function and plotted with `seqmsplot()`. Figure 7 shows how such modal state sequences are displayed. The height of the bar at each position is proportional to the frequency of the displayed state at that position. The number of occurrences of the modal state sequence is also displayed. Since the shown sequences of modal states do not belong to the sequence dataset, the number of occurrences is 0 for both considered groups.

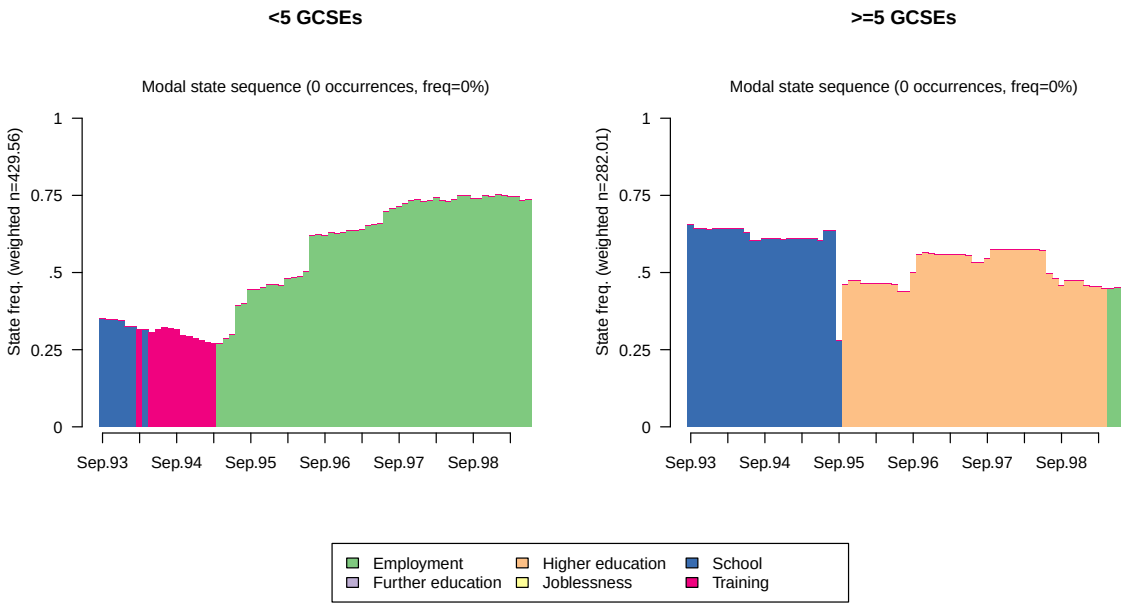

Figure 7: Modal state sequence by end of compulsory school qualification group.

Transversal entropy of state distributions

In addition to the state distribution, the `seqstatd()` function provides for each position in the sequence the number of valid states and the Shannon entropy of the transversal state distribution. Shannon's entropy, also known as the *entropy index*, has been applied to social science data by, for instance, Billari (2001a) and Fussell (2005). Letting p_i denote the proportion of cases in state i at the considered position, the entropy is

$$h(p_1, \ldots, p_a) = -\sum_{i=1}^{a} p_i \log(p_i)$$

where a is the size of the alphabet. The entropy is 0 when all cases are in the same state and is maximal when we have the same proportion of cases in each state. The entropy can be seen as a measure of the diversity of states observed at the considered position.

Plotting the transversal entropies can be useful to find out how the diversity of states evolves along the time axis. We plot transversal entropies with `seqHtplot()`. Figure 8 shows the curves by end of compulsory school qualification group. For the first group, the entropy of the state distributions noticeably decreases at the end of the follow-up period. This is a consequence of the increasing proportion of youngsters entering into full employment (Figure 6). For the second group, the entropy index slightly increases at the end of the considered period, which may be explained by the emergence of two balanced subgroups, namely those who continue higher education and those who enter into employment.

7. Individual sequence characteristics

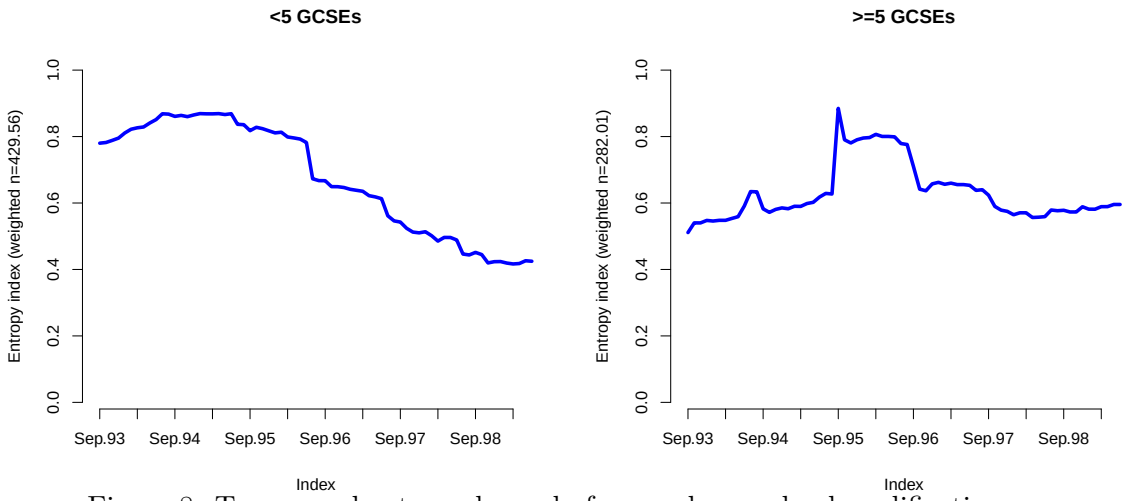

Figure 8: Transversal entropy by end of compulsory school qualification group.

We focus now on the characterization and summarization of longitudinal characteristics of individual sequences. Essentially, the aim is to define measures that inform on how each sequence is constituted; i.e., on whether it takes a simple or more complex form.

The interpretation of complexity indexes will depend indeed on the context. Consider for instance the number of transitions—changes of state—in a sequence. When looking at work trajectories, for example, sequences with numerous transitions may correspond to unusual disrupted trajectories. In other contexts such as family formation, sequences with fewer transitions may indicate that an individual failed to pass through the usual stages of the family formation (leaving the parental home, cohabitation with a partner, birth of one or more children, etc...).

In the SPS form (see Section 3) a state sequence is represented as an ordered list of successive distinct states with their associated durations; i.e., as a sequence of couples (x_j, t_j) where x_j is a state and t_j its duration. This suggests that we can distinguish characteristics of the state sequencing—the distinct successive states (DSS)—from those of the durations.[8] We first examine two indicators of the state sequencing and one based on the durations. More synthetic measures are addressed in Section 7.2.

7.1. Unidimensional indicators

Number of transitions

Perhaps the simplest indicator is the number of transitions in the sequence; i.e., the number of state changes. The number of transitions in a sequence x is readily obtained from the length $\ell_d(x)$ of its DSS sequence. It is $\ell_d(x) - 1$. We get the number of transitions for each sequence of state sequence object with `seqtransn()`.

Number of subsequences

[8]The two pieces of information can be extracted separately with `seqdss()` and `seqdur()`.

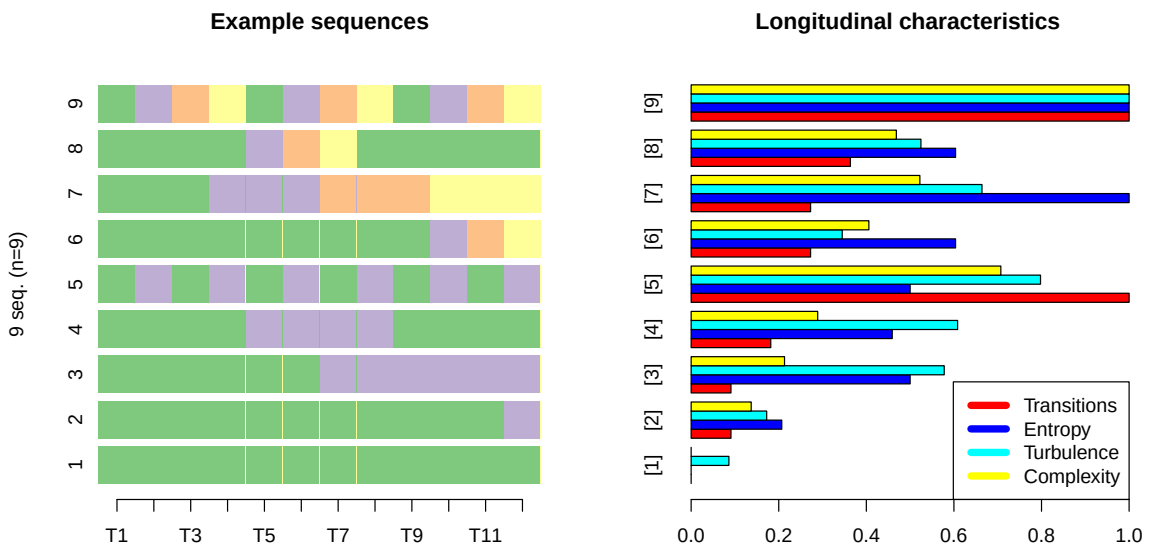

Figure 9: Example sequences ($\ell = 12$, $a = 4$) and normalized values of complexity measures.

The number $\phi(x)$ of subsequences that can be extracted from the DSS sequence provides also useful information on the sequencing of the states. This measure is returned by the `seqsubsn()` function. It is used in the turbulence measure presented below. A subsequence y of x is composed of elements of x occurring in the same order than in x.

The maximal number of subsequences is reached only for a sequence made of the repetition of the alphabet. In Figure 9, for example, sequences 5 and 9 have the maximal number of transitions, while the number of subsequences is maximal for sequence 9 only.

Within sequence entropy

Regarding the durations, we consider the total time spent in each state; i.e., in case of multiple spells in a same state, the sum of the lengths of these spells. For example, in (EM,4)-(TR,2)-(EM,64), the first sequence in the object *mvad.seq*, there are two spells in state EM with respective durations 4 and 64. Hence, the time spent in state EM is 68 months, as shown by the output of the `seqistatd()` function:

```
R> seqistatd(mvad.seq[1:4, ])
```

```
  EM FE HE JL SC TR
1 68  0  0  0  0  2
2  0 36 34  0  0  0
3 10 34  0  2  0 24
4 14  0  0  9  0 47
```

The total time spent in each state characterizes the state distribution within a sequence. The entropy of this distribution can be seen as a measure of the diversity of its states. We call it *within* or *longitudinal entropy* to distinguish from the transversal entropy considered in Section 6.2 on page 20.

The `seqient()` function returns the longitudinal Shannon entropies; i.e., for each sequence the value of

$$h(\pi_1, \ldots, \pi_a) = -\sum_{i=1}^{a} \pi_i \log \pi_i$$

where a is the size of the alphabet and π_i the proportion of occurrences of the ith state in the considered sequence. When the state remains the same during the whole sequence, the entropy equals 0, while the maximum entropy is reached when the same time is spent inside the sequence in each possible element of the alphabet. By default the entropy is normalized by dividing the value of $h(\pi_1, \ldots, \pi_s)$ by its theoretical maximum, $\log a$.[9] Figure 9 helps to get a more concrete idea of what the entropy measures. We see that the within-sequence entropy does not account for the state order in the sequence. For instance, sequences 7 and 9 have the same maximal normalized entropy of 1.

7.2. Composite complexity measures

The previous measures are based either on the sequencing or on the durations. We look now at composite measures that account simultaneously for those two aspects.

Turbulence

The *turbulence* $T(x)$ of a sequence x is a composite measure proposed by Elzinga (Elzinga and Liefbroer 2007) that accounts for the number $\phi(x)$ of distinct subsequences of the DSS sequence and the variance $s_t^2(x)$ of the consecutive times t_j spent in the $\ell_d(x)$ distinct states. The formula is

$$T(x) = \log_2 \left(\phi(x) \, \frac{s_{t,max}^2(x) + 1}{s_t^2(x) + 1} \right)$$

where $s_{t,max}^2(x)$ is the maximum value that $s_t^2(x)$ can take given the total duration $\ell(x) = \sum_j t_j$ of that sequence. This maximum is $s_{t,max}^2(x) = \big(\ell_d(x) - 1\big)\big(1 - \bar{t}(x)\big)^2$ where $\bar{t}(x)$ is the mean consecutive time spent in the distinct states.

From a prediction point of view, the higher the differences in state durations and hence the higher their variance, the less uncertain the sequence. In that sense, small duration variance indicates high complexity.

The vector containing the turbulences of the sequences in a sequence object is obtained with the `seqST()` function.

Complexity index

The *complexity index*, introduced in Gabadinho, Ritschard, Studer, and Müller (2010), is a composite measure that combines the number of transitions in the sequence with the longi-

[9] The latter is indeed an upper bound for the maximal entropy. It is exactly the maximal possible entropy only when the sequence length is a multiple of the alphabet size. Use `norm=FALSE` to disable normalization.

tudinal entropy. It reads

$$C(x) = \sqrt{\frac{(\ell_d(x) - 1)}{(\ell(x) - 1)} \frac{h(x)}{h_{max}}}$$

where h_{max} is the theoretical maximum value of the entropy given the alphabet; i.e., $h_{max} = \log a$. We get the vector of complexity indexes with the `seqici()` function.

The minimum value of 0 can only be reached by a sequence with a single distinct state; i.e., with no transition and an entropy of 0. $C(x)$ reaches its maximum 1 if and only if the sequence x is such that i) x contains each of the states in the alphabet, ii) the same time $\ell(x)/a$ is spent in each state, and iii) the number of transitions is $\ell(x) - 1$.

Complexity index versus turbulence

It is instructive to look at how the turbulence and complexity indexes behave for the examples in Figure 9. The turbulence produces significantly higher values for sequences 3 and 4, which have a rather low 'sequencing' complexity but a null variance of their state durations. Indeed, this variance does not account for states that are not visited, which tends to give high turbulence values to seemingly simple sequences such as sequence 3 with two spells of same length and hence a null variance of their durations. Similarly, the turbulence exceeds the complexity index for sequences 3, 4, 5, and 7, which all have a zero variance in duration and, hence, a relatively high turbulence value. The longitudinal entropy that intervenes in the complexity index is another way of looking at the time spent in the states. It accounts, on its side, for all states, including the nonvisited ones, and, therefore, discriminates clearly between the sequences with zero duration variance.

8. Measuring sequence (dis)similarity

We examine now how we can measure the dissimilarity between two state sequences. As we will see in Section 9, once we have pairwise dissimilarities we will be able to run many types of powerful classical and specific statistical analysis methods on sequence data.

Many sequence dissimilarity measures have been proposed in the literature, of which the most popular in social sciences is the optimal matching (OM) edit distance. **TraMineR** offers a general `seqdist()` function that can compute the OM dissimilarity as well as a set of other dissimilarity measures. Table 6 lists the available methods and their required parameters. The `seqdist()` function can output the matrix of pairwise dissimilarities or the vector of distances to a provided reference sequence. We can also compute multichannel dissimilarities (Pollock 2007) with the `seqdistmc()` function.

Dissimilarity measures can be classified into measures based on the count of matching attributes and those defined as the (minimal) cost of transforming one sequence into the other. Another interesting distinction is between those that make position-wise comparisons; i.e., that do not allow shifting a sequence or part of it, and those accounting for similar shifted patterns (see Table 6). Without shift, $x = ABAB$ and $y = BABA$ are very distant, while they are quite similar if we shift y by just one position.

Distance	Method	Position-wise	Additional arguments
Count of common attributes			
Simple Hamming	`HAM`	Yes	
Longest Common Prefix	`LCP`	Yes	
Longest Common Suffix	`RLCP`	Yes	
Longest Common Subsequence	`LCS`	No	
Edit distances			
Optimal Matching	`OM`	No	Insertion/deletion costs (`indel`) and substitution costs matrix (`sm`)
Hamming	`HAM`	Yes	substitution costs matrix (`sm`)
Dynamic Hamming	`DHD`	Yes	substitution costs matrix (`sm`)

Table 6: List of available metrics for computing distances with the `seqdist()` function.

8.1. Dissimilarities based on counts of common attributes

Let $A(x, y)$ be a count of common attributes between sequences x and y. It is a proximity measure since the higher the counts, the closer the sequences. We derive a dissimilarity measure from it through the following general formula

$$d(x, y) = A(x, x) + A(y, y) - 2A(x, y) \tag{1}$$

where $d(x, y)$ is the distance between objects x and y. The dissimilarity is maximal when $A(x, y) = 0$; i.e., when the two sequences have no common attribute. It is zero when the sequences are identical, in which case we have $A(x, y) = A(x, x) = A(y, y)$. Let us briefly describe the implemented count-based dissimilarity measures.

The simple *Hamming distance* (Hamming 1950) is the number of positions at which two sequences of equal length differ. It can equivalently be defined as $\ell - A_H(x, y)$, with $\ell = |x| = |y|$ the common sequence length and $A_H(x, y)$ the number of matching positions.[10] We get the Hamming distance with Equation (1) by using $A_H(x, y)/2$ as proximity measure.

We obtain another simple distance measure by using the length $A_P(x, y)$ of the *longest common prefix* (LCP) between two sequences; i.e., by counting the number of successive common positions starting from the beginning of the sequences[11] (see for instance Elzinga 2007b). The reversed longest common prefix (RLCP) or *longest common suffix* is similar to the LCP but looks for the common elements from the end rather than from the beginning of the sequences.

Another implemented metric is based on the length $A_S(x, y)$ of the *longest common subsequence* (LCS).[12] Notice that consecutive states in the common subsequence are not necessarily consecutive in the compared sequences. For example, the length of the LCS between sequences 1 and 3 of `mvad.seq` (see page 11 and Figure 2 on page 13) is 12 and we get 59 between sequences 2 and 5.

Quite obviously, we can only have $A_S(x, y) \geq A_P(x, y)$; i.e., the length of the LCS cannot be smaller than the length of the LCP, and hence the LCS distance cannot be greater than

[10] This latter quantity is returned by the `seqmpos()` function.
[11] This length is returned by the `seqLLCP()` function.
[12] The length of the LCS is returned by the `seqLLCS()` function.

the LCP distance. We have also $A_S(x, y) \geq A_H(x, y)$. When compared with metrics based on position-wise counts such as the simple Hamming and the LCP distances, the LCS metric reduces distances by accounting for non-aligned matches; i.e., position-shifted similarities.

8.2. Edit distances

An edit distance is defined as the minimal cost of transforming one sequence into the other. This cost depends, indeed, on the allowed transforming operations and their individual costs. Basically, two types of operations are considered: i) the substitution of one element by an other one, and ii) the *indel*; i.e., the insertion or deletion of an element, which generates a one-position shift of all the elements on its right. The generalized Hamming (HAM) and dynamic Hamming distances (DHD) (Lesnard 2006) accept only substitutions and hence no shift. The former assumes position-independent substitution costs while the second allows for position-dependent costs. The Optimal Matching (OM) distance, first considered by Levenshtein (1966) and popularized in the social sciences by Abbott (Abbott and Forrest 1986), accounts for both operations.

Setting indels and substitution costs

Usually the indel cost is set as a constant independent of the concerned position and state. Setting a high indel cost relatively to substitution costs favors substitutions while low values favor indels. We can prohibit shifts by setting the indel cost sufficiently high.[13]

Substitution costs are generally organized in matrix form. A three-dimensional matrix is necessary in the case of position varying costs as used, for instance, by the DHD metric. In the time invariant case, the substitution-cost matrix is a square symmetrical matrix of dimension $a \times a$, where a is the number of distinct states in the alphabet. The element (i, j) in the matrix is the cost of substituting state s_i with state s_j. The user can either specify its own substitution-cost matrix,[14] or compute one by means of the `seqsubm()` function with option `method = "CONSTANT"` or `method = "TRATE"`. With `"CONSTANT"`, all costs are set as the user provided `cval` constant (2 by default). With `"TRATE"`, the costs are determined from the estimated transition rates as

$$2 - p(s_i \mid s_j) - p(s_j \mid s_i)$$

where $p(s_i \mid s_j)$ is the probability of observing state s_i at time $t + 1$ given that state s_j has been observed at time t (see page 17). The idea is to set a high cost when changes between s_i and s_j are seldom observed and lower cost when they are frequent.

Here is how we get the time-invariant transition-rate-based substitution cost matrix for the `mvad` data:

```
R> scost <- seqsubm(mvad.seq, method = "TRATE")
R> round(scost, 3)
```

[13] It is sufficient, for prohibiting indels, to set the indel cost higher than $\frac{\ell}{2} \times \max(sm)$, where ℓ is the common sequence length and $\max(sm)$ is the highest substitution cost.

[14] When this substitution-cost matrix does not respect the triangle inequality, dissimilarities based on it may also fail to respect this inequality (see Studer *et al.* 2011).

```
      EM->  FE->  HE->  JL->  SC->  TR->
EM-> 0.000 1.971 1.987 1.957 1.988 1.961
FE-> 1.971 0.000 1.993 1.977 1.991 1.993
HE-> 1.987 1.993 0.000 1.997 1.981 1.999
JL-> 1.957 1.977 1.997 0.000 1.992 1.976
SC-> 1.988 1.991 1.981 1.992 0.000 1.995
TR-> 1.961 1.993 1.999 1.976 1.995 0.000
```

The minimum cost is 0 for the substitution of each state by itself, and the maximum is less than 2; i.e., the value that we would get for a transition not observed in the data. In accordance with what we observed in the transition rate matrix (page 18), we get the lowest costs for substituting EM (employment) to JL (joblessness) or TR (training). Remember, however, that—unlike transition rates—the substitution costs are symmetric and hence we have the lowest cost for changing JL or TR into EM.

Implemented edit distances

Generalized Hamming (HAM) and *Dynamic Hamming* (DHD) dissimilarities are intended for sequences of equal lengths only. The former generalizes the basic Hamming distance by allowing for state dependent substitution costs. Indeed, the count of nonmatching positions is the cost of substituting a state at each position when all costs are set to 1. DHD is the extension proposed by Lesnard (2006) to account for time-varying costs. For the *mvad* data set, the flexibility in substitution costs allowed by the DHD metric has only a limited impact as can be seen in Figure 10.

In addition to substitutions, *Optimal Matching* (OM) allows also insertions/deletions. It thus applies to sequences of unequal lengths. Since the cost of the transformation may vary with the order of the successive indels and substitutions, OM is defined as the minimal cost—in terms of insertions, deletions and substitutions—of transforming one sequence into the other one. The cost minimization is achieved through dynamic programming, the algorithm implemented in **TraMineR** being essentially that of Needleman and Wunsch (1970) with standard optimizations.

The 712×712 pairwise distance matrix for our `mvad` data computed by using the transition-rate-based costs and an indel cost of 1 is obtained with the command:

```
R> mvad.om <- seqdist(mvad.seq, method = "OM", indel = 1, sm = scost)
```

The `mvad.om` distance matrix requires only 3.87 megabytes memory space. However the number n of sequences in the data can be an important issue when computing dissimilarity matrices since both the computing time and the size of the resulting matrix increase exponentially with n. If necessary, we can divide the size by 2 by requesting only the upper triangle of the matrix with the `full.matrix=FALSE` argument. Most R functions accept the resulting upper-triangle objects as dissimilarity argument.

Comparing dissimilarity measures

Choosing a dissimilarity measure and setting substitution and indel costs is an important step in sequence analysis. Though popular in social sciences, distances based on such costs have

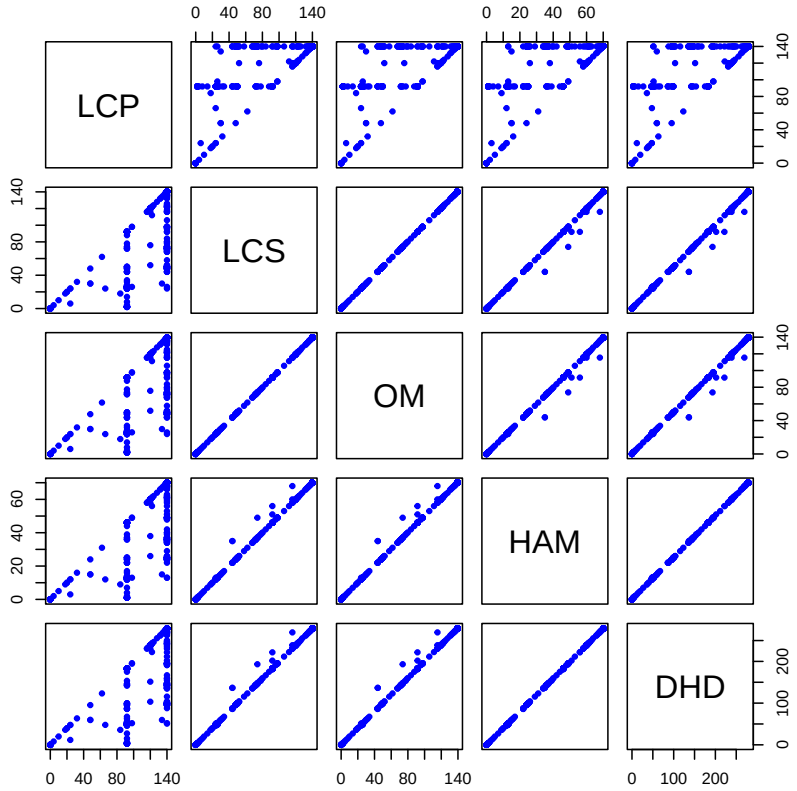

Figure 10: Distances to the most frequent sequence obtained with various metrics, `mvad` data.

raised questions in the literature (see for instance Dijkstra and Taris 1995; Wu 2000; Elzinga 2007b). The meaning of the substitution costs, their required symmetry and the sensitivity of the results to the chosen values have been pointed out as important issues. More recently, the meaning of indels was also addressed (Hollister 2009; Lesnard 2010). Favoring insertions and deletions reduces the importance of time shifts in the comparison, while favoring substitutions gives more importance to position-wise similarities.

Comparing the results obtained with various settings can also be useful for selecting the appropriate measure. Figure 10 compares the discussed dissimilarity measures for the distance to the most frequent sequence on the `mvad` data. We observe that, apart from the LCP metric, the measures yield very similar results. The few significant differences between HAM (or DHD) and LCS (or OM) illustrate how LCS and OM reduce dissimilarity by allowing for shifts in the comparison of the sequences. The mean difference between OM, obtained with costs derived from substitution rates, and LCS is only 0.4% of the maximal distance. The largest difference is 0.63%. These small differences are a consequence of low transition rates which lead to substitution costs comprised between 1.96 and 2; i.e., close to 2. With a constant substitution cost of 2 and an indel cost equal to 1, OM is just LCS (Elzinga 2007b).

8.3. Normalized distances

When dealing with sequences of different lengths, we may want to normalize the distances to account for these differences. More specifically, the aim of normalization is to relativize distances such that a dissimilarity of say 10 between sequences of length 100 becomes less important than a dissimilarity of 10 between sequences of length 5. While the maximal distance between a pair of sequences depends on their length, normalization aims at setting it to 1 or, at least, to a value that does not depend on the lengths.[15]

With `seqdist()` we control normalization by means of the `norm` argument. When setting it to `TRUE`, the normalization applied is determined by the selected metric. For LCP, RLCP and LCS, we apply Elzinga (2007b)'s normalization. It works as follows. Letting $A(x, y)$ be the (non normalized) proximity measure, we first normalize this similarity

$$s_A(x, y) = \frac{A(x, y)}{\sqrt{A(x, x)A(y, y)}}$$

The normalized distance is, then, just the complement to 1 of the normalized similarity.

$$D_A(x, y) = 1 - s_A(x, y)$$

which gives values comprised between 0 and 1.

For the OM distance, as well as for HAM and DHD, we apply Abbott's normalization, which consists of dividing the distance by the length of the longest of the two sequences

$$D_{OM}(x, y) = \frac{d_{OM}(x, y)}{\max\{|x|, |y|\}} \ .$$

It results that for OM with an indel cost of 1 and a constant substitution cost of 2, the maximal normalized OM distance is 2. Though OM is in this latter case equivalent to LCS, their normalized values differ.

We can also force the normalization method by specifying either `"gmean"` for Elzinga's normalization or `"maxlength"` for Abbott's solution. Alternatively, we can use `"maxdist"`, which consists of dividing each distance by its maximal theoretical value. For LCP and LCS distances, the maximal possible value is the sum $\ell_x + \ell_y$ of the lengths of the two sequences x and y, for HAM it is the length ℓ of the sequences, while the maximum theoretical OM distance is

$$D_{max} = \min(\ell_x, \ell_y) \cdot \min\left(2c_I, \max(S)\right) + c_I|\ell_x - \ell_y|$$

where $c_I > 0$ is the indel cost, $\max(S)$ the greatest substitution cost and $|\ell_x - \ell_y|$ the absolute value of the difference in lengths of the two sequences. With the unit indel cost and the `scost` transition-rate-based substitution cost matrix, this yields 139.94 for the `mvad` data. This is very close from twice the sequence length; i.e., $2 \cdot 70 = 140$.

It is worth mentioning that the triangle inequality property of the original distance may in some cases be lost through the `"maxlength"` and `"maxdist"` normalizations.

9. Dissimilarity based sequence analysis

[15] Normalization is not intended to account for the differences in length of the sequences between which we are measuring the dissimilarity.

Beside information on the similarity between any pair of sequences, a distance matrix opens access to many classical statistical and data analysis tools. It permits for instance to extract representative sequences such as medoids, to run any clustering technique based on pairwise dissimilarities and to apply multidimensional scaling. It even permits to compute pseudo-variances and run ANOVA-like analyses as explained in Studer *et al.* (2011). We demonstrate in this section how the `mvad.om` dissimilarity (distance) matrix obtained with the command shown page 27 can be exploited for further statistical analysis.

9.1. Representative sequences

A major concern when analyzing sets of categorical sequences is to find useful ways of summarizing them. Possible solutions could be to determine some central or typical sequence such as the modal—most frequent—sequence or the medoid—most central—sequence. However, such solutions are of limited interest since they provide usually only a too rough idea of the main patterns in the set. A more general approach consists in finding sets of representatives and **TraMineR** provides the versatile generic `seqrep()` function for extracting such sets from the dissimilarity matrix. The function allows control over the amount of information that the representative set should convey. The sets returned by `seqrep()` exhibit, thus, the key features of the whole set they are extracted from, which proves useful, for example, when labeling clusters of sequences.

The principle of the search algorithm (Gabadinho, Ritschard, Studer, and Müller 2011) is to sort the sequences according to a representativeness criterion[16] and to remove the redundancy by browsing the sorted sequences. The redundancy threshold is set as a percentage (10% by default) of the maximum theoretical dissimilarity D_{max} between two sequences and the representative set will thus not contain any pair of sequences that are nearer each other than this threshold. The size of the representative set can be controlled by fixing either the minimal expected `coverage` of the representative set or the number `nrep` of representatives.

The *coverage* of a representative sequence is the percentage of sequences that are in its neighborhood; i.e., the number of sequences with a distance to the representative less than a selected threshold.[17] The *total coverage* of the representative set corresponds to the percentage of the n original sequences that have a representative in their neighborhood. A series of other individual and global measures to evaluate the quality of the obtained representatives is also computed.

The list of representative sequences is obtained by printing the outcome of `seqrep()` and we get the quality measures with the `summary()` method. The `seqrplot()` function generates representative sequence plots.

Example 1: Medoid and the centrality criterion

A first simple example [18] of a representative sequence is the *medoid* of a set of sequences. The medoid is the most central object; i.e., the one with minimal sum of distances to all other objects in the set (Kaufman and Rousseeuw 2005). It is a special case of representative se-

[16] See the reference manual for the different criteria that can be used for sorting the original sequences.

[17] We suggest setting the threshold as a given percentage of the maximum theoretical distance between two sequences.

[18] Results for representative sequences differ from the ones published in the original JSS article. This is due to the use of weights, which was not implemented in the TraMineR version (1.8) the article is based on.

quence obtained by selecting the centrality sorting criterion (`criterion="dist"`) and setting the size of the representative set to 1 (`nrep=1`).

```
R> medoid <- seqrep(mvad.seq, diss = mvad.om, criterion = "dist",
+       nrep = 1)
R> print(medoid, format = "SPS")
```

```
 [>] criterion: dist
 [>] 711.57 sequence(s) in the original data set
 [>] 1 representative sequence(s)
 [>] overall quality: -60.84

    Sequence
[1] (SC,24)-(FE,8)-(EM,38)
```

Example 2: Representative set and the neighborhood density criterion

The medoid of a set of sequences usually yields poor coverage. To increase coverage we should allow for more than one representative. When seeking for more than one representative, an initial sort of the sequences according to the density of their neighborhood yields better results. Neighborhood density is, therefore, the default criterion used by `seqrep()`.

The command below finds and plots the representative set that, with a neighborhood radius of 10% (default `pradius` value), covers at least 25% (default `coverage` value) of the sequences in each of the two *gcse5eq* groups:

```
R> seqrplot(mvad.seq, group = mvad$gcse5eq, diss = mvad.om, border = NA)
```

In the resulting plot (Figure 11) the selected representative sequences are plotted bottom-up according to their representativeness score with bar width proportional to the number of sequences assigned to them. At the top of the plot, two parallel series of symbols standing each for a representative are displayed horizontally on a scale ranging from 0 to the maximal theoretical distance D_{max}. The location of the symbol associated with the representative, r_i, indicates on axis A the discrepancy within the subset R_i of sequences assigned to r_i and on axis B the mean distance to the representative.

We learn from the plots that respectively five and one representatives are necessary for each of the two groups to achieve the 25% coverage and that the actual coverage is 29% in both cases.

9.2. Clustering sequences

Clustering is an exploratory data analysis method aimed at finding automatically homogeneous groups or clusters in the data (Kaufman and Rousseeuw 2005). In life course studies (e.g., McVicar and Anyadike-Danes 2002; Widmer and Ritschard 2009), the method has typically been used in combination with OM distances to identify distinct groups of sequences with similar patterns; that is, to define a typology of sequences.

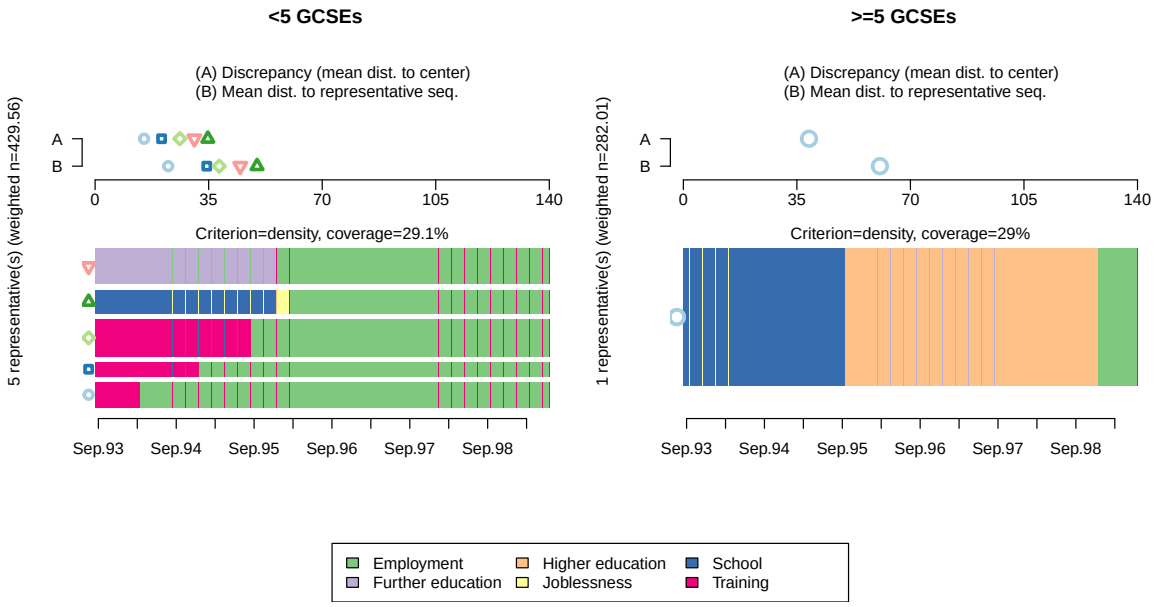

Figure 11: Representative sequences by end of compulsory school qualification group.

We already showed, in Section 2, how we can make a cluster analysis of sequences using the **cluster** library (Maechler, Rousseeuw, Struyf, and Hubert 2005). We used `agnes()` to make a hierarchical clustering with the Ward method, but `pam()` (partitioning around medoids) or `diana()` (divisive analysis), for example, could also be used. The four clusters solution was retained after examining the dendrogram (Figure 12) of the clustering tree obtained with:

```
R> plot(clusterward, which.plots = 2, labels = FALSE)
```

Figure 13 on page 34 obtained with the command below shows the representative sequences by cluster, complementing the plots of the transversal state distributions shown in Figure 1 page 7. The threshold for the coverage of the representative set is set to 35% using the `coverage=.35` argument.

```
R> seqrplot(mvad.seq, group = cl4.lab, diss = mvad.om, coverage = 0.35,
+       border = NA)
```

Looking at these two figures helps interpreting and labeling the clusters. They show that clustering from the OM distances identifies four distinct patterns of school to work transitions. In the first cluster the trajectories are clearly oriented toward early transition to employment, with, in some cases, a spell of training. The second cluster is dominated by trajectories containing a spell of school or further education followed by higher education. Cluster 3 corresponds to slow transition to employment with first an important spell of further education. In the last cluster, the transitions from school to work are more chaotic with frequent spells of training and joblessness.

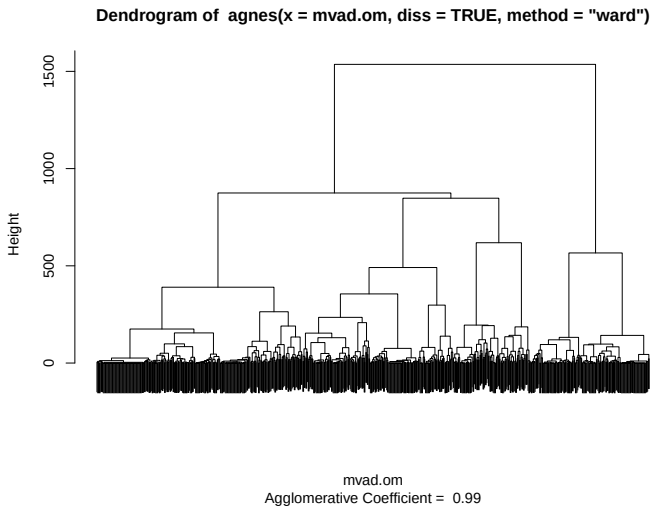

Figure 12: Hierarchical sequence clustering from the OM distances, Ward method.

	Clust 1	Sig	Clust 2	Sig	Clust 3	Sig	Clust 4	Sig
(Constant)	0.812	0.152	0.069	0.000	0.437	0.000	0.194	0.000
Male	1.616	0.005	0.938	0.767	0.653	0.014	0.951	0.822
Father unemployed	0.753	0.204	0.625	0.166	1.022	0.926	2.015	0.006
Good end cs grade	0.170	0.000	14.987	0.000	1.150	0.432	0.356	0.000

Table 7: Odds Ratios for cluster memberships.

After having labeled the clusters, a common further step is to examine how the cluster membership depends on covariates by means of logistic regressions. We can, for instance, investigate the profiles of the youngsters belonging to the last cluster with:

```
R> mb4 <- (cl4.lab == "Cluster 4")
R> glm.cl4 <- glm(mb4 ~ male + funemp + gcse5eq, data = mvad,
+       family = "binomial")
```

Proceeding the same way for the other clusters and taking the exponential of the regression coefficients, we get the odds ratios shown in Table 7. We learn from those figures that, among others, men are significantly more present in Cluster 1 and women in Cluster 3, and that the chances to follow the trajectory pattern with higher education depicted by Cluster 2 are multiplied by about 15 for those with good results at the end of compulsory school.

10. Conclusion

TraMineR is a unique toolbox for categorical sequential data that offers, in a unified environment, a series of methods that could previously only be found in separate programs. It also provides original measures, plots and methods of analysis that were developed by the authors.

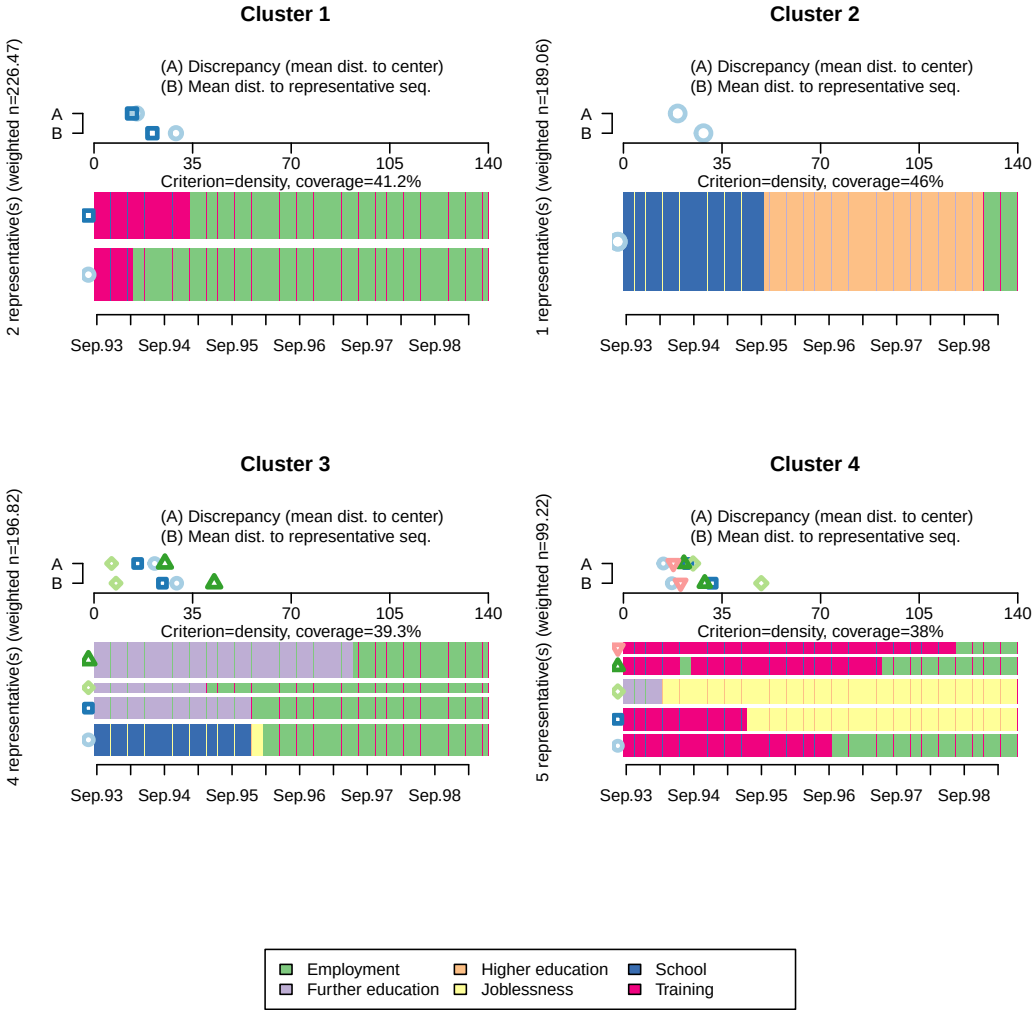

Figure 13: Plots of representative sequences by cluster.

Besides the material for state sequences described in this article, the package includes specific tools for mining event sequences (Ritschard *et al.* 2008; Müller, Studer, Ritschard, and Gabadinho 2010; Studer, Müller, Ritschard, and Gabadinho 2010), for studying the discrepancy of a set of sequences through ANOVA-like analyses and regression trees (Studer *et al.* 2011). In addition, it proposes utilities such as the conversion to and from various sequence formats (Ritschard *et al.* 2009) that was briefly addressed in Section 3.

This rich set of features, in combination with other specialized R packages, allows the user to run all steps of a complete sequence analysis in a single, free, powerful and multi-platform environment. It is also worth mentioning that development regarding state sequences continues. Related papers and information on current developments can be found on the package's Web page `http://mephisto.unige.ch/traminer`. We encourage users to submit their feature requests using our dedicated tool at `http://r-forge.r-project.org/projects/traminer`.

Acknowledgments

The development of the **TraMineR** package and this article were realized within a research project supported by the Swiss National Science Foundation under grants 113998 and 122230.

References

Aassve A, Billari F, Piccarreta R (2007). "Strings of Adulthood: A Sequence Analysis of Young British Women's Work-Family Trajectories." *European Journal of Population*, **23**(3), 369–388. URL http://dx.doi.org/10.1007/s10680-007-9134-6.

Abbott A (1997). "Optimize." URL http://home.uchicago.edu/~aabbott/om.html.

Abbott A, Forrest J (1986). "Optimal Matching Methods for Historical Sequences." *Journal of Interdisciplinary History*, **16**, 471–494.

Abbott A, Tsay A (2000). "Sequence Analysis and Optimal Matching Methods in Sociology, Review and Prospect." *Sociological Methods and Research*, **29**(1), 3–33.

Berchtold A, Raftery AE (2002). "The Mixture Transition Distribution Model for High-Order Markov Chains and Non-Gaussian Time Series." *Statistical Science*, **17**(3), 328–356.

Billari FC (2001a). "The Analysis of Early Life Courses: Complex Description of the Transition to Adulthood." *Journal of Population Research*, **18**(2), 119–142.

Billari FC (2001b). "Sequence Analysis in Demographic Research." *Canadian Studies in Population*, **28**(2), 439–458. Special Issue on Longitudinal Methodology.

Billari FC, Fürnkranz J, Prskawetz A (2006). "Timing, Sequencing, and Quantum of Life Course Events: A Machine Learning Approach." *European Journal of Population*, **22**(1), 37–65.

Blossfeld HP, Golsch K, Rohwer G (2007). *Event History Analysis with Stata*. Lawrence Erlbaum, Mahwah NJ.

Brzinsky-Fay C, Kohler U, Luniak M (2006). "Sequence Analysis with Stata." *The Stata Journal*, **6**(4), 435–460.

Deville JC, Saporta G (1983). "Correspondence analysis with an extension towards nominal time series." *Journal of Econometrics*, **22**, 169–189.

Dijkstra W, Taris T (1995). "Measuring the Agreement between Sequences." *Sociological Methods and Research*, **24**(2), 214–231.

Elzinga CH (2007a). *CHESA 2.1 User Manual*. Vrije Universiteit, Amsterdam. URL http://home.fsw.vu.nl/ch.elzinga/.

Elzinga CH (2007b). "Sequence Analysis: Metric Representations of Categorical Time Series." *Manuscript*, Department of Social Science Research Methods, Vrije Universiteit, Amsterdam. URL http://home.fsw.vu.nl/ch.elzinga/.

Elzinga CH, Liefbroer AC (2007). "De-standardization of Family-Life Trajectories of Young Adults: A Cross-National Comparison Using Sequence Analysis." *European Journal of Population*, **23**, 225–250. doi:10.1007/s10680-007-9133-7.

Fussell E (2005). "Measuring the Early Adult Life Course in Mexico: An Application of the Entropy Index." In R Macmillan (ed.), "The Structure of the Life Course: Standardized? Individualized? Differentiated?", Advances in Life Course Research, Vol. 9, pp. 91–122. Elsevier, Amsterdam.

Gabadinho A, Ritschard G, Studer M, Müller NS (2009). "Mining Sequence Data in R with the **TraMineR** package: A User's Guide." *Technical report*, Department of Econometrics and Laboratory of Demography, University of Geneva, Geneva. URL http://mephisto.unige.ch/traminer/.

Gabadinho A, Ritschard G, Studer M, Müller NS (2010). "Indice de complexité pour le tri et la comparaison de séquences catégorielles." *Revue des nouvelles technologies de l'information RNTI*, **E-19**, 61–66.

Gabadinho A, Ritschard G, Studer M, Müller NS (2011). "Extracting and Rendering Representative Sequences." In A Fred, JLG Dietz, K Liu, J Filipe (eds.), "Knowledge Discovery, Knowledge Engineering and Knowledge Management," volume 128 of *Communications in Computer and Information Science (CCIS)*, pp. 94–106. Springer-Verlag.

Hamming RW (1950). "Error Detecting and Error Correcting Codes." *Bell System Techical Journal*, **29**, 147–160.

Hollister M (2009). "Is Optimal Matching Suboptimal?" *Sociological Methods Research*, **38**(2), 235–264. doi:10.1177/0049124109346164.

Hosmer DW, Lemeshow S (1999). *Applied Survival Analysis, Regression Modeling of Time to Event Data*. John Wiley & Sons, New York.

Kaufman L, Rousseeuw PJ (2005). *Finding Groups in Data*. John Wiley & Sons, Hoboken.

Lesnard L (2006). "Optimal Matching and Social Sciences." *Série des Documents de Travail du CREST 2006-01*, Institut National de la Statistique et des Etudes Economiques, Paris.

Lesnard L (2010). "Setting Cost in Optimal Matching to Uncover Contemporaneous Socio-Temporal Patterns." *Sociological Methods and Research*, **38**, 389–419. doi:10.1177/0049124110362526.

Levenshtein V (1966). "Binary Codes Capable of Correcting Deletions, Insertions, and Reversals." *Soviet Physics Doklady*, **10**, 707–710.

Maechler M, Rousseeuw P, Struyf A, Hubert M (2005). "Package '**cluster**': Cluster Analysis Basics and Extensions." *Reference manual*, R-project, CRAN. URL http://CRAN.R-project.org/package=cluster.

Mayer KU, Tuma N (1990). "Life course research and event history analysis: An overview." In KU Mayer, N Tuma (eds.), "Event History Analysis in Life Course Research," pp. 3–20. WI: University of Wisconsin Press, Madison.

McVicar D, Anyadike-Danes M (2002). "Predicting Successful and Unsuccessful Transitions from School to Work Using Sequence Methods." *Journal of the Royal Statistical Society A*, **165**(2), 317–334.

Müller NS, Studer M, Ritschard G, Gabadinho A (2010). "Extraction de règles d'association séquentielle l'aide de modèles de durée." *Revue des nouvelles technologies de l'information RNTI*, **E-19**, 25–36.

Needleman S, Wunsch C (1970). "A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins." *Journal of Molecular Biology*, **48**, 443–453.

Neuwirth E (2007). ***RColorBrewer**: ColorBrewer Palettes*. R package version 1.0-2, URL http://CRAN.R-project.org/package=RColorBrewer.

Pollock G (2007). "Holistic Trajectories: A Study of Combined Employment, Housing and Family Careers by Using Multiple-Sequence Analysis." *Journal of the Royal Statistical Society A*, **170**(1), 167–183. doi:10.1111/j.1467-985X.2006.00450.x.

R Development Core Team (2011). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.r-project.org.

Ritschard G, Gabadinho A, Müller NS, Studer M (2008). "Mining Event Histories: A Social Science Perspective." *International Journal of Data Mining, Modelling and Management*, **1**(1), 68–90. doi:10.1504/IJDMMM.2008.022538.

Ritschard G, Gabadinho A, Studer M, Müller NS (2009). "Converting between Various Sequence Representations." In Z Ras, A Dardzinska (eds.), "Advances in Data Management," volume 223 of *Studies in Computational Intelligence*, pp. 155–175. Springer-Verlag, Berlin. doi:10.1007/978-3-642-02190-9_8.

Rohwer G, Ptter U (2002). "TDA: Transition Data Analysis." *Software*, Ruhr-Universität Bochum, Fakultät für Sozialwissenschaften, Bochum. URL http://www.ruhr-uni-bochum.de/tda.html.

Scherer S (2001). "Early Career Patterns: A Comparison of Great Britain and West Germany." *European Sociological Review*, **17**(2), 119–144.

Studer M, Müller NS, Ritschard G, Gabadinho A (2010). "Classer, discriminer et visualiser des séquences d'événements." *Revue des nouvelles technologies de l'information RNTI*, **E-19**, 37–48.

Studer M, Ritschard G, Gabadinho A, Müller NS (2011). "Discrepancy Analysis of State Sequences." *Sociological Methods and Research*. In press.

Widmer E, Ritschard G (2009). "The De-Standardization of the Life Course: Are Men and Women Equal?" *Advances in Life Course Research*, **14**(1-2), 28–39. doi:10.1016/j.alcr.2009.04.001.

Wu LL (2000). "Some Comments on 'Sequence Analysis and Optimal Matching Methods in Sociology: Review and Prospect'." *Sociological Methods Research*, **29**(1), 41–64. doi:10.1177/0049124100029001003.

Yamaguchi K (1991). *Event History Analysis*. ASRM 28. Sage, Newbury Park and London.

Affiliation:

Alexis Gabadinho, Gilbert Ritschard, Nicolas S. Müller, Matthias Studer
Institute for Demographic and Life Course Studies
University of Geneva
CH-1211 Geneva 4, Switzerland
E-mail: alexis.gabadinho@unige.ch
URL: http://mephisto.unige.ch/