Check for updates

The Stata Journal (2017)
17, Number 3, pp. 546–572

SADI: Sequence analysis tools for Stata

Brendan Halpin
Department of Sociology
University of Limerick
Limerick, Ireland
brendan.halpin@ul.ie

Abstract. The `SADI` package provides tools for sequence analysis, which focuses on the similarity and dissimilarity between categorical time series such as life-course trajectories. `SADI`'s main components are tools to calculate intersequence distances using several different algorithms, including the optimal matching algorithm, but it also includes utilities to graph, summarize, and manage sequence data. It provides similar functionality to the R package `TraMineR` and the Stata package `SQ` but is substantially faster than the latter.

Keywords: st0486, SADI, ari, sdchronogram, combinadd, combinprep, corrsqm, cumuldur, sddiscrep, dynhamming, sdentropy, sdhamming, sdhollister, maketrpr, metricp, nspells, oma, omav, permtab, sdstripe, trans2subs, trprgr, twed, sequence analysis tools

1 Sequence analysis tools for the social sciences

In this article, I present the `SADI` package (sequence analysis distance measures), which offers a set of tools for carrying out sequence analysis on categorical time-series data such as life-course trajectories. It offers functionality that overlaps extensively with the `TraMineR` package for R (Gabadinho et al. 2009) and the `SQ` package for Stata (Brzinsky-Fay, Kohler, and Luniak 2006).

Sequence analysis is an approach to longitudinal (or otherwise sequential) social science data that treats sequences as wholes. It is an alternative to models that focus on transition rates or other analytical summaries (such as hazard rate models). It usually proceeds by defining a pairwise sequence dissimilarity score and uses this score in procedures such as cluster analysis. Sequence analysis has been used in the social sciences since the 1980s (Abbott and Forrest 1986) but has developed extensively in the past two decades (see, for example, Halpin [2013] for a general summary of the literature, Cornwell [2015] for a didactic approach, and Blanchard, Bühlmann, and Gauthier [2014] for a summary of recent developments). Sequence analysis is also seen in geography (for example, Bargeman, Joh, and Timmermans [2002]) and surgical research (for example, Forestier et al. [2012]).

Typically, research in sequence analysis depends on the optimal matching algorithm (OMA), where the distance between any two sequences is defined as the cost of the cheapest set of edits (insertions, deletions, and substitutions) that will turn one sequence into the other. The matrix of all pairwise distances may then be used to generate a data-driven classification via cluster analysis (for example, Halpin and Chan [1998]), or se-

© 2017 StataCorp LLC

quences may be classified by proximity to ideal types (for example, Martin, Schoon, and Ross [2008]). Hypotheses about destandardization of the life course may be addressed by looking at average distances within cohorts (for example, Elzinga and Liefbroer [2007]). Dyadic analyses look at pairs of sequences such as siblings' fertility histories (for example, Raab et al. [2014]).

2 The SADI toolkit

`SADI` provides many sequence analysis tools along with distance measures, including "optimal matching" distance (perhaps the most popular sequence analysis distance). `SADI` also has utilities for graphing sequence-related data, for summarizing sequences, and for handling sequences in general.

The main alternatives to `SADI` are the Stata `SQ` package (Brzinsky-Fay, Kohler, and Luniak 2006) and the R package `TraMineR` (Gabadinho et al. 2009). `SADI` provides some tools that are not in `SQ` and is much faster for some important functions.[1] `TraMineR` is an attractive environment for those working in R, but `SADI` makes it possible to do a lot in a Stata environment and has distance measures that are not in `TraMineR`.

Because some of the distance measures are relatively intensive to calculate, they are implemented as C plugins rather than pure Stata or Mata code. This means that they are available only for Windows, Linux, and Mac OS, 32- and 64-bit. If you would like to compile them for another platform, see appendix B.

I will now summarize the functionality offered by `SADI` and then present detailed worked examples.

3 SADI functionality

`SADI` consists of several utilities: distance measures that define distances or dissimilarities between pairs of sequences, graphical summaries, tools for cluster analysis, utilities creating variables summarizing the sequences, tools for analyzing the distance matrices, and some general helper functions.

See appendix A (or help files, once installed) for fuller descriptions of each command.

3.1 Distance measures

Distance measure commands take sequences (defined as consecutive runs of variables representing the same categorical state space over time) and generate matrices of pairwise distances, using different definitions of distance (or similarity).

1. The latest version of `SQ` provides the option of using `SADI` functionality for some operations, giving it access to the greater speed and extra distance measures.

- `combinadd` calculates intersequence distances using Elzinga's duration-weighted subsequence counting.
- `dynhamming` calculates intersequence distances using dynamic Hamming distance.
- `sdhamming` calculates intersequence distances using Hamming distance.
- `sdhollister` calculates intersequence distances using Hollister's localized optimal matching.
- `oma` calculates intersequence distances using the Needleman–Wunsch (or optimal matching) algorithm.
- `omav` calculates intersequence distances via the duration-compensated Needleman–Wunsch algorithm.
- `twed` calculates intersequence distances using time-warp edit distance.

Many distance measures in `SADI` are discussed in detail in Halpin (2012, 2014) and Studer and Ritschard (2016).

3.2 Graphical summaries

- `sdchronogram` graphs the time-dependent state distribution.
- `trprgr` graphs the time series of transition rates between states.

3.3 Cluster related tools

Sequence analysis often proceeds by subjecting the pairwise distance matrix to cluster analysis. `SADI` provides several cluster-analysis related tools:

- `ari` calculates the adjusted Rand index comparing two cluster solutions.
- `permtab` compares two cluster solutions by permuting columns to maximize agreement as defined by Cohen's Kappa.

3.4 Summary variables

Many tools create variables summarizing aspects of the sequences:

- `cumuldur` calculates cumulated duration in states of a sequence.
- `sdentropy` calculates the Shannon entropy of a sequence.
- `nspells` calculates the number of spells in a sequence.
- `sdstripe` creates a single string variable representing the sequence.

3.5 Distance matrix tools

Several tools focus on the matrix of pairwise distances:

- `corrsqm` calculates the correlation between the lower triangle of two symmetric matrices.
- `sddiscrep` calculates Studer et al.'s (2011) discrepancy measure with respect to a grouping variable, carrying out a pseudo–analysis of variance.
- `metricp` tests a symmetric matrix of pairwise distances for the triangle inequality.

3.6 Helper utilities

- `combinprep` transforms sequences from wide calendar format to wide spell format, preparing the data for `combinadd`.
- `maketrpr` creates a matrix containing transition rates from sequences (used by `trprgr` and `dynhamming`).
- `trans2subs` creates a substitution matrix based on observed transitions.

4 Installation

Several commands in the package depend on the `mm_expand()` Mata function found in Ben Jann's (2005) `moremata` package, installed as follows:

```
. ssc install moremata
```

I also recommend installing the `SQ` package for sequence analysis, not least for its effective implementation of index plots:

```
. ssc install sq
```

5 Data requirements

Sequence analysis works with linear structures, usually but not necessarily longitudinal in time, that are discrete in both the longitudinal dimension (as measured) and the state space. Typically, each element represents a time period or event in sequential order and contains an observation in a categorical state space. A typical example is monthly labor market status, but any ordered sequence of observations in a categorical state space will qualify. More examples are sequences of coded utterances in a conversation, or steps in a dance.

`SADI` expects sequences to be represented by a consecutive run of variables, where the categories are numbered from 1 up to the number of categories. Thus, each case contains

a complete sequence in wide format. Missing values are not accommodated unless missing is treated as a category in its own right (see Halpin [2016b] for an approach to multiple imputation suited to sequence data). Sequences of different length should start at time-point 1 and have a variable indicating their length.

6 Worked example

In this section, the functionality of `SADI` is presented by example. All the steps presented are included in a Stata do-file, available as part of the `SADI` package as an ancillary file.

6.1 Quick start

The `SADI` package may be installed from the *Stata Journal* website using the commands

```
. net sj 17-3 st0486
. net install st0486
```

The following commands will install its dependencies:

```
. ssc install moremata
. ssc install sq
```

These commands download the example data and do-file to carry out the examples in the following pages and run the following do-file:

```
. net get st0486
. do distances.do
```

6.2 Data

We use data from McVicar and Anyadike-Danes (2002) and set up a substitution matrix (that is, a description of distances within the state space, in this case defined a priori by the authors). The data consist of 72 monthly observations (`state1` to `state72`), in a 6-element state space, on the transition from school to work (respectively, employment, further education, higher education, secondary education, training, and unemployment).

```
. use mvad
. sort id
. set matsize 4000
. matrix mvdanes = (0,1,1,2,1,3 \
>                   1,0,1,2,1,3 \
>                   1,1,0,2,1,2 \
>                   2,2,2,0,1,1 \
>                   1,1,1,1,0,2 \
>                   3,3,2,1,2,0 )
```

6.3 Calculating distances

Sequence analysis proceeds by calculating distances between pairs of sequences, typically generating matrices of distances between all pairs (see immediately below) or distances to reference sequences (see below).

Most distance measures work with the sequences as strings of state variables and have a relatively consistent set of options. The following code creates six _N × _N pairwise distance matrices using six different distance measures:

```
. oma         state1-state72, subsmat(mvdanes) pwdist(omd) length(72) indel(1.5)
  (output omitted)
. omav        state1-state72, subsmat(mvdanes) pwdist(omv) length(72) indel(1.5)
  (output omitted)
. sdhollister state1-state72, subsmat(mvdanes) pwdist(hol) length(72)
> timecost(0.5) localcost(0.5)
  (output omitted)
. twed        state1-state72, subsmat(mvdanes) pwdist(twd) length(72) lambda(0.5)
> nu(0.04)
  (output omitted)
. sdhamming   state1-state72, subsmat(mvdanes) pwdist(ham)
  (output omitted)
. dynhamming  state1-state72,                  pwdist(dyn)
  (output omitted)
```

The commands start with a variable list that defines the sequence, and then have different options. Where relevant, `subsmat()` provides the substitution cost or state-space distance information. The mandatory `pwdist()` option names the matrix in which the pairwise distances are returned. Where it is possible to compare sequences of different length, `length()` specifies the length either as a constant or as a variable (in these data, sequences are all the same length, but all the above distances, except the two Hamming distances, can accommodate sequences of variable length). Other options are command-specific.

The measure `omav` is described in Halpin (2010), `sdhollister` in Hollister (2009), `dynhamming` in Lesnard (2008), and `twed` in Marteau (2009, 2008) and Halpin (2014).

X/t

The X/t measure, a duration-weighted, spell-oriented version of Elzinga's number of matching subsequences similarity measure, is calculated with `combinadd`. It is described in Elzinga (2007) and discussed in Halpin (2014). It counts the number of subsequences that are present in both sequences, weighting by duration. Because it works with spells rather than calendar-format data, we need to restructure the data into a wide spell format (consisting of a state variable and a length variable for each spell). The `combinprep` command does the restructuring, and `combinadd` calculates the distances. We need to know the maximum number of spells in the data, which is returned as `r(maxspells)` by `combinprep`.

```
. preserve
. combinprep, state(state) length(len) idvar(id) nspells(nspells)
  (output omitted)
. local spmax = r(maxspells)
. combinadd state1-len`spmax´, pwsim(xts) nspells(nspells) nstates(6) rtype(d)
  (output omitted)
. restore
```

Distance to reference sequences

Where theoretically or empirically derived reference or "ideal-type" sequences are available, it can be useful (and quicker) to calculate distances to the reference sequences rather than all pairwise distances. This is available for `oma`, `dynhamming`, and `twed` (and for Hamming distance by means of using `oma` with a sufficiently high insertion or deletion cost). If `oma`, `dynhamming`, or `twed` is given the `ref(#)` option, a $\_N \times \#$ matrix of distances is created, where the distances are to the first $\#$ sequences in the data (that is, the reference sequences need to be added to the top of the dataset; see ancillary file `distances.do` for a more detailed example).

```
. oma        state1-state72, subsmat(mvdanes) pwdist(ham3) ref(3) length(72)
> indel(999)
  (output omitted)
. oma        state1-state72, subsmat(mvdanes) pwdist(omd3) ref(3) length(72)
> indel(1.5)
  (output omitted)
. twed       state1-state72, subsmat(mvdanes) pwdist(twd3) ref(3) length(72)
> lambda(0.5) nu(0.04)
  (output omitted)
. dynhamming state1-state72,                  pwdist(dyn3) ref(3)
```

Data-driven substitution matrix

The substitution cost matrix used by many of the distance measures defines similarity and dissimilarity between the states of the state space. Sometimes, researchers use theory or prior information to generate these values. Some researchers prefer to use the data to generate it from transition rates (note that `dynhamming` does this automatically but using time-varying transition rates). This may or may not be a good idea; there is a homology between transition rate matrices and substitution matrices, but substitutions and transitions are orthogonal concepts.

The `trans2subs` command creates a matrix of the transition rate-based distances. Typically, transitions will occur much less often than once per time unit, so the diagonal will be heavily populated. Thus, the off-diagonal transition rates will be low, and distances will have low variability. If we exclude the diagonal, we get distances with greater variability.

Distances are defined as $2 - p_{ij} - p_{ji}$ where $p_{ij} = n_{ij}/n_{i+}$.

To calculate the transition rates, you must enter the data in long format:

```
. preserve
. reshape long state, i(id) j(m)
(note: j = 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
> 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52
> 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72)
Data                               wide   ->   long
-----------------------------------------------------------------------------
Number of obs.                      712   ->   51264
Number of variables                  86   ->      16
j variable (72 values)                    ->   m
xij variables:
              state1 state2 ... state72   ->   state
-----------------------------------------------------------------------------
. trans2subs state, idvar(id) subsmat(tpr1)
(712 missing values generated)
Generating transition-driven substitution matrix
. matrix list tpr1
symmetric tpr1[6,6]
          c1        c2        c3        c4        c5        c6
r1         0
r2  1.147539         0
r3  1.064734  1.849958         0
r4  1.643575  1.757525  1.671111         0
r5  1.182927  1.844291      1.96   1.90181         0
r6  1.207729  1.525335  1.831594  1.803575  1.608297         0
. trans2subs state, idvar(id) subsmat(tpr2) diagincl
(712 missing values generated)
Generating transition-driven substitution matrix
. matrix list tpr2
symmetric tpr2[6,6]
          c1        c2        c3        c4        c5        c6
r1         0
r2  1.967601         0
r3   1.98727  1.993341         0
r4  1.984684  1.987531  1.982969         0
r5  1.959993  1.992045  1.999488  1.994867         0
r6  1.951231   1.96336  1.996033  1.985649  1.972029         0
. restore
```

Note that the substitution costs have a much smaller range when the diagonal is included.

We can then calculate OMA distances using the transition-derived substitution costs, excluding the diagonal:

```
. oma state1-state72, subsmat(tpr1) pwdist(tpr) length(72) indel(1.5)
```

(*output omitted*)

6.4 Examining distance matrices

Comparing distances

Between different distance measures and different parameterizations, or substitution costs, we now have eight pairwise distance matrices. The simplest way to compare them is correlation. The `corrsqm` command reports the Pearson correlation between the lower triangles of two square (symmetric) matrices, optionally excluding the diagonal (which, for distance matrices, is filled with zeros for all measures).

```
. foreach dist in dyn ham twd hol omv xts tpr {
  2.   corrsqm omd `dist´, nodiag
  3. }
VECH correlation between omd and dyn: 0.7915
Diagonal suppressed
VECH correlation between omd and ham: 0.9856
Diagonal suppressed
VECH correlation between omd and twd: 0.8065
Diagonal suppressed
VECH correlation between omd and hol: 0.9898
Diagonal suppressed
VECH correlation between omd and omv: 0.9197
Diagonal suppressed
VECH correlation between omd and xts: 0.1135
Diagonal suppressed
VECH correlation between omd and tpr: 0.7701
Diagonal suppressed
```

Note the very high correlation with OMA of the Hamming and Hollister measure, the very low correlation of the combinatorial X/t measure, and the relatively big difference between OMA with the original substitution cost matrix and OMA with the transition-rate based matrix.

The triangle inequality

For many of these distance matrices' uses, they must imply a metric space. This requires, inter alia, that the distances obey the triangle inequality: for all A and B, there is no C such that $d(A,B) > d(A,C) + d(C,B)$. The `omav` and `sdhollister` distances do not fulfill this requirement (see Halpin [2014]).

```
. foreach dist in dyn ham twd hol omv xts tpr {
  2.   metricp `dist´
  3. }
Matrix dyn is consistent with a metric space
Matrix ham is consistent with a metric space
Matrix twd is consistent with a metric space
Shorter route exists between seq   1 and seq 210 -- 148.000 > 147.000
Shorter route exists between seq   2 and seq  12 -- 112.000 > 111.000
Shorter route exists between seq   2 and seq  28 -- 110.000 > 109.500
Shorter route exists between seq   2 and seq  56 -- 67.000 > 66.500
Shorter route exists between seq   2 and seq  64 -- 50.500 > 50.000
Shorter route exists between seq   2 and seq  71 -- 98.000 > 97.000
Shorter route exists between seq   2 and seq  77 -- 64.000 > 63.500
Shorter route exists between seq   2 and seq  81 -- 65.000 > 64.500
Shorter route exists between seq   2 and seq 113 -- 172.000 > 171.000
Shorter route exists between seq   2 and seq 142 -- 35.000 > 34.000
Matrix hol is NOT consistent with a metric space
Shorter route exists between seq   1 and seq   2 -- 11.580 > 9.428
Shorter route exists between seq   1 and seq   3 -- 10.313 > 8.721
Shorter route exists between seq   1 and seq   5 -- 11.846 > 9.428
Shorter route exists between seq   1 and seq   6 -- 6.982 > 5.167
Shorter route exists between seq   1 and seq   7 -- 6.707 > 4.714
Shorter route exists between seq   1 and seq   8 -- 4.693 > 3.064
Shorter route exists between seq   1 and seq   9 -- 4.994 > 3.543
Shorter route exists between seq   1 and seq  10 -- 10.826 > 7.778
Shorter route exists between seq   1 and seq  11 -- 9.478 > 7.660
Shorter route exists between seq   1 and seq  12 -- 11.706 > 10.017
Matrix omv is NOT consistent with a metric space
Matrix xts is consistent with a metric space
Matrix tpr is consistent with a metric space
```

The `sdhollister` and `omv` distance matrices are not metric and are hence of limited value. Only the first 10 exceptions are printed unless the `detailed` option is given.

6.5 Cluster analysis

Often sequence analysis proceeds by conducting cluster analysis on the pairwise distance matrix. Here we do it for the `oma` and `twed` distances, generating cluster solutions with 8 and 12 clusters in each case by using Stata's built-in `clustermat` and `cluster generate` commands:

```
. clustermat wards omd, name(oma) add
. cluster generate o=groups(8 12)
. clustermat wards twd, name(twd) add
. cluster generate t=groups(8 12)
```

Comparing cluster solutions

We can compare the cluster solutions for the two measures in several ways. Clusterings are "unlabeled classifications" in that clusters can be identified only by reference to the cases they contain. In this sense, a cluster in a clustering based on one distance matrix is "the same" or similar to a cluster in a clustering based on another matrix only to the extent that they contain (mostly) the same cases.

The adjusted Rand index (Hubert and Arabie 1985; Vinh, Epps, and Bailey 2009) reflects agreement defined as the extent to which the members of a pair of cases, if in the same cluster in one solution, are in the same cluster in the other:

```
. ari o8 t8
Adjusted Rand Index:  0.5977
```

The `permtab` command cross-tabulates two solutions, permuting the values of one to maximize the agreement. The permuted classification can be saved as a new variable:

```
. permtab o8 t8, gen(pt8) tables
Tabulating raw data:
```

	t8					
o8	1	2	3	4	5	Total
1	92	1	0	0	0	93
2	41	96	0	2	0	139
3	0	0	0	4	0	62
4	0	4	16	123	0	146
5	11	19	9	2	39	93
6	0	0	0	0	2	30
7	2	5	28	1	4	47
8	0	0	14	0	1	102
Total	146	125	67	132	46	712

	t8			
o8	6	7	8	Total
1	0	0	0	93
2	0	0	0	139
3	0	57	1	62
4	0	2	1	146
5	13	0	0	93
6	28	0	0	30
7	0	0	7	47
8	0	0	87	102
Total	41	59	96	712

```
Calculating permutations:
Kappa max: 0.7346
Permutation vector:
```

	1	2	3	4	5	6	7	8
1	1	2	7	4	5	6	3	8

```
Permuted table:
```

	1	2	3	4	5	6	7	8
1	92	1	0	0	0	0	0	0
2	41	96	0	2	0	0	0	0
3	0	0	57	4	0	0	0	1
4	0	4	2	123	0	0	16	1
5	11	19	0	2	39	13	9	0
6	0	0	0	0	2	28	0	0
7	2	5	0	1	4	0	28	7
8	0	0	0	0	1	0	14	87

```
Original table:
          1     2     3     4     5     6     7     8
  1    92     1     0     0     0     0     0     0
  2    41    96     0     2     0     0     0     0
  3     0     0     0     4     0     0    57     1
  4     0     4    16   123     0     0     2     1
  5    11    19     9     2    39    13     0     0
  6     0     0     0     0     2    28     0     0
  7     2     5    28     1     4     0     0     7
  8     0     0    14     0     1     0     0    87
Permuted column variable generated from t8: pt8
```

The permutation seeks to maximize Cohen's κ as an index of agreement (Reilly, Wang, and Rutherford 2005), and reports the $\kappa_{\max}$ to be 0.7346.

Permutation is simple but expensive if there are many categories. For 12 clusters, permutation takes $9 \times 10 \times 11 \times 12 = 11880$ times as long as for 8. To deal with this, `permtab` with the `algorithm(ga)` option yields an approximate-best permutation using a genetic algorithm:

```
. permtab o12 t12, gen(pt18) algorithm(ga)
```
(*output omitted*)

The $\kappa_{\max}$ and adjusted Rand index will generally be in close agreement. While `permtab` is slower than `ari`, it has the advantage of creating a new variable containing the permuted version of the second cluster solution that best matches the first solution.

Discrepancy

Studer et al.'s (2011) discrepancy measure brings a pseudo–analysis of variance perspective to distance matrices (Studer et al. 2011). If we partition the matrix using a cluster solution, or a preexisting observed characteristic, we can compare the average distance to the center of the partition with the average distance to the overall center and generate a pseudo-R^2 measure. This uses the distance to the center as an analogue of sums of squared deviations, and where the distances are squared Euclidean, it will generate results that are numerically equivalent.

The approach uses bootstrapping to generate p-values, and increasing the `niter()` option from the default 100 iterations increases precision. See Studer et al. (2011) for more detail.

```
. sddiscrep o8, distmat(omd) idvar(id)
Discrepancy based R2 and F, 100 permutations for p-value
```

	pseudo R2	pseudo F	p-value
o8	.5310534	113.891	.01

```
. sddiscrep o12, distmat(omd) idvar(id)
Discrepancy based R2 and F, 100 permutations for p-value
             | pseudo R2   pseudo F    p-value
-------------+------------------------------
         o12 |  .5990087   95.06125        .01
. sddiscrep grammar, distmat(omd) idvar(id)
Discrepancy based R2 and F, 100 permutations for p-value
             | pseudo R2   pseudo F    p-value
-------------+------------------------------
     grammar |  .8342287   3573.008        .01
. sddiscrep grammar, distmat(omd) idvar(id) niter(1000)
Discrepancy based R2 and F, 1000 permutations for p-value
             | pseudo R2   pseudo F    p-value
-------------+------------------------------
     grammar |  .8342287   3573.008       .005
```

Note that if a cluster solution is used as the classifying variable, the discrepancy pseudo-F is the same as the Caliński–Harabasz cluster stopping-rule pseudo-F. See `ssc describe calinski` and Halpin (2016a) for more information about using cluster stopping rules with pairwise distance matrices.[2]

6.6 Summarizing sequences and clusters

String representations of sequences

We can create string representations of sequences, which makes it much easier to get a visual overview of the data and allows searching for patterns. By default, `sdstripe` uses capital letters in alphabetical order to represent the sequential states, but the `symbols()` option allows any sequence of characters to be used (here `"EFHSTU"` matches the states).

```
. sdstripe state1-state72, generate(seqstr) symbols("EFHSTU")
Creating long string representation
. list seqstr in 1/5, clean
                                                                        seqstr
  1.  TTEEEETTEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
  2.  UUFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH
  3.  UUTTTTTTTTTTTTTTTTTTTTTTTFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEEEEEEEEEEUU
  4.  TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTEEEEEEEEEEEEEUUUUUUUUU
  5.  UUFFFFFFFFFFFFFFFFFFFFFFFFFFHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH
```

Stata's `regex` system makes it easy to search for patterns in these representations. For instance, `count if regexm(seqstr, "^E+$")` will count sequences 100% in employment, `count if regexm(seqstr, "U[^U]")` will count sequences where we observe an exit from unemployment, and `count if regexm(seqstr, "U[^U]U+[^U]U")` will

2. Stata's built-in `cluster stop` command does not work on pairwise distance matrices. The `calinski` and `dudahart` commands, available on Statistical Software Components archive and described in Halpin (2016a), will work correctly on such matrices.

count sequences that experience at least three separate spells of unemployment. Regular expressions can describe quite complex and general patterns.

`sdstripe` can also generate "condensed" sequence representations based on the spell structure:

```
. sdstripe state1-state72, generate(seqstrxt) symbols("EFHSTU") xt
> xtspellsep("/") xtdursep(":")
Creating condensed string representation
. list seqstrxt in 1/5, clean
                    seqstrxt
  1.         T:2/E:4/T:2/E:64
  2.            U:2/F:36/H:34
  3.   U:2/T:24/F:34/E:10/U:2
  4.            T:49/E:14/U:9
  5.            U:2/F:25/H:45
```

Medoids: Typical sequences

We can characterize clusters in many ways (see below for graphics, `sdchronogram` and `sqindexplot`). One way is to pick a "medoid", the sequence nearest the center of the cluster. The `sddiscrep` command has an option to save this distance as a variable, which allows us to identify the medoid. The medoids are all simple and quite distinct:

```
. sddiscrep o8, distmat(omd) idvar(id) dcg(dx) niter(1) // niter(1) because
> p-value not needed
Discrepancy based R2 and F, 1 permutations for p-value
```

	pseudo R2	pseudo F	p-value
o8	.5310534	113.891	1

o8	N(dx)	min(dx)	mean(dx)	max(dx)
1	93	1.241993	4.231125	16.60758
2	139	3.936442	11.70385	31.09472
3	62	3.302549	11.11681	33.62513
4	146	6.059251	16.38595	57.24418
5	93	26.78911	37.66251	65.05792
6	30	8.887777	19.84556	38.58778
7	47	7.227705	18.77229	50.20643
8	102	3.599865	14.70406	67.12928

```
. sort o8 dx
. by o8: gen medoid = _n==1
```

```
. list o8 dx seqstrxt if medoid, clean
       o8         dx        seqstrxt
  1.    1   1.241993            E:72
 94.    2   3.936442       T:24/E:48
233.    3   3.302549       F:27/H:45
295.    4   6.059251   S:2/F:34/E:36
441.    5   26.78911   T:49/E:14/U:9
534.    6   8.887777       T:20/U:52
564.    7   7.227705       S:26/E:46
611.    8   3.599865       S:26/H:46
. sort id
```

Cumulated duration

The sequencewise total duration in each state is also an interesting summary:

```
. cumuldur state1-state72, cd(dur) nstates(6)
```

Even though cumulated duration discards all order information, it differentiates the clusters very strongly (see below).

Entropy

We can look at the entropy of cumulated duration. The `sdentropy` command calculates a simple measure of Shannon entropy (maximal if all states are equally likely, minimal if only one state is visited):

```
. // first drop the cumulated duration variables as the
. // sdentropy command will recreate these
. drop dur1-dur6
. sdentropy state1-state72, gen(ent) cd(dur) nstates(6)
```

Because this measure completely ignores order, it is not an entirely appropriate measure of sequence complexity. Nonetheless, entropy levels differ greatly by cluster (see below).

Number of spells

The total number of spells in a sequence is a measure of its volatility.

```
. nspells state1-state72, gen(nsp)
```

Cumulated duration, entropy, and spells all differ strongly across the clusters:

```
. table o8, c(mean dur1 mean dur2 mean dur3) format(%5.2f)
```

o8	mean(dur1)	mean(dur2)	mean(dur3)
1	0.94	0.04	0.00
2	0.68	0.03	0.00
3	0.09	0.38	0.50
4	0.48	0.43	0.02
5	0.34	0.11	0.00
6	0.07	0.05	0.00
7	0.46	0.07	0.05
8	0.06	0.06	0.46

```
. table o8, c(mean dur4 mean dur5 mean dur6) format(%5.2f)
```

o8	mean(dur4)	mean(dur5)	mean(dur6)
1	0.00	0.01	0.01
2	0.02	0.25	0.02
3	0.01	0.00	0.02
4	0.01	0.03	0.03
5	0.01	0.28	0.25
6	0.06	0.11	0.70
7	0.30	0.05	0.07
8	0.37	0.01	0.03

6.7 Graphics

Two key graphics are associated with sequence analysis: the state-distribution plot (or chronogram) and the index plot. I also present a representation of the time-structure of transition rates.

Chronogram

The chronogram represents the distribution of states at each time unit, hiding individual continuity but yielding a more digestible summary:

```
. sdchronogram state*, by(o8, legend(off)) name(chronogram, replace)
(0 observations deleted)
Creating chronogram data
Drawing chronogram
```

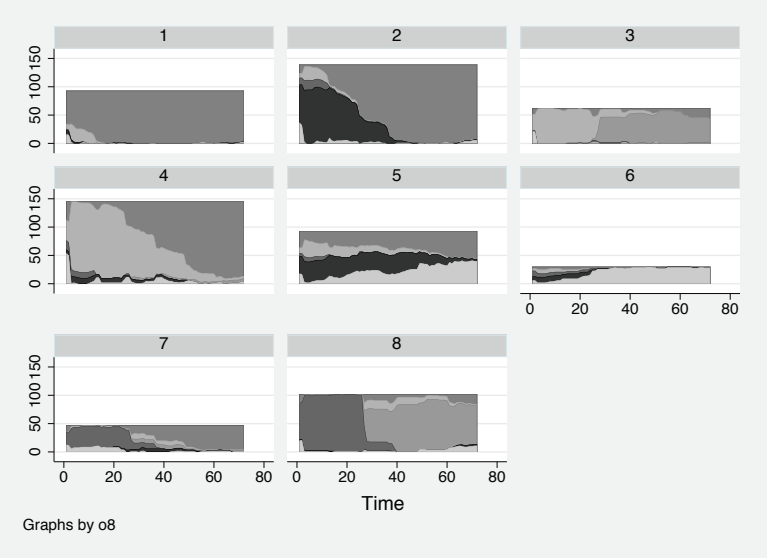

Figure 1. Chronogram, by eight-cluster OMA solution

Index plot

The index plot plots each sequence as a line and thus reproduces the sequence data in full. The `sqindexplot` command from the `SQ` package implements this very well, so it has not been reimplemented for `SADI`; we use the `SQ` implementation here.

Type `ssc install sq` if necessary.

To make the full sequence data visually digestible, you must group and order them carefully. If we plot by cluster, the order within clusters is critical. My preference is to generate a maximal clustering (as many clusters as distinct sequences). This allows us to order sequences within clusters so that subcluster structure is preserved (such that the sequences are in dendrogram order, thus showing the structure of subclusters within clusters). It makes clustered index plots more readable and less dependent on cutting at an arbitrary number of clusters.

```
. cluster generate o999 = groups(750), name(oma) ties(fewer)
```

SQ wants sequence data in long format and to be sqset:

```
. preserve
. reshape long state, i(id) j(m)
  (output omitted)
. sqset state id m
  (output omitted)
. sqindexplot, by(o8, note("") legend(off)) order(o999) name(indexplot, replace)
. restore
```

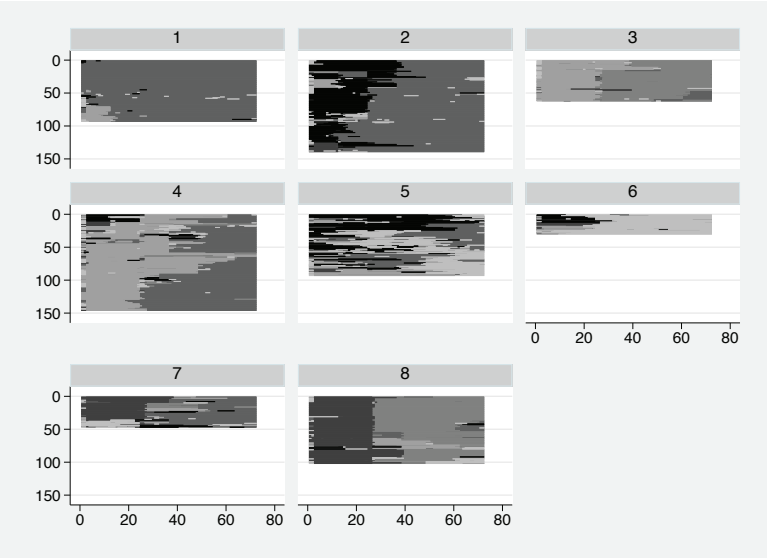

Figure 2. Index plot, by eight-cluster OMA solution

Transition pattern graph

The trprgr command creates a composite graphic, with a column of graphs (chronograms) representing (in this example) the 6 states over time, and a 6×6 grid of line graphs representing the transition rates between states over time. The grid is analogous to a transition table, but with graphed time series rather than transition rates for each origin–destination combination.

trprgr gives an alternative view of the sequences, highlighting the evolution of transition rates over time rather than intersequence distance.

```
. trprgr state*, gmax(485)
```
(*output omitted*)

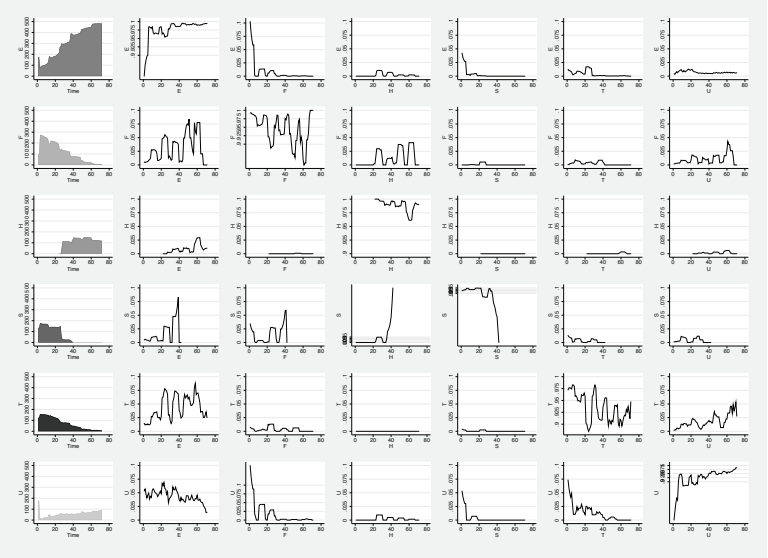

Figure 3. Transition pattern

By design, this command shows transition rates on the diagonal on the range 0.9–1.0 and those off the diagonal on the range 0.0–0.1, but in practice, these ranges are often exceeded. See the `ceiling()` and `floor()` options.

maketrpr

The `maketrpr` command generates the matrix of transition rates used by `dynhamming` and `trprgr`, using `tssmooth` to average over a moving window of successive transitions. It may be of interest to inspect these data in matrix form as much as in the `trprgr` graph.

```
. maketrpr state*, matrix(mkt) ma(5)
```

For m categories and t time points, this creates a $(t-1) \times m \times m$ matrix, where each successive $m \times m$ panel represents a time-specific pattern of transitions (smoothed by `tssmooth`). In this example, there are no early observations in state 3 (higher education), so its exit rate is undefined:

```
. matlist mkt[1..6,.]
```

	__0000071	__0000072	__0000073	__0000074	__0000075	__0000076
r1	.8853028	.0684874	0	.0297906	.0101547	.0062645
r2	.0054704	.9890236	0	0	.0024511	.0030549
r3	.	.	.	.	.	.
r4	.0050338	.0228499	0	.9606053	.009611	.0019001
r5	.0129851	.0049919	0	.0026247	.9770244	.0023739
r6	.0473975	.1041914	0	.0354406	.0490166	.7639539

```
. matlist mkt[25..30,.]
```

	__0000071	__0000072	__0000073	__0000074	__0000075	__0000076
r25	.9244586	.0410925	0	.0188743	.0070362	.0085383
r26	.0071804	.9871928	0	0	.0026261	.0030007
r27	.	.	.	.	.	.
r28	.0030203	.0137099	0	.975197	.0057666	.0023062
r29	.0136273	.0029951	0	.0015748	.9771932	.0046096
r30	.0481792	.0625148	0	.0212644	.0368173	.8312242

7 References

Abbott, A., and J. Forrest. 1986. Optimal matching methods for historical sequences. *Journal of Interdisciplinary History* 16: 471–494.

Bargeman, B., C.-H. Joh, and H. Timmermans. 2002. Vacation behavior using a sequence alignment method. *Annals of Tourism Research* 29: 320–337.

Blanchard, P., F. Bühlmann, and J.-A. Gauthier, eds. 2014. *Advances in Sequence Analysis: Theory, Method, Applications*. Berlin: Springer.

Brzinsky-Fay, C., U. Kohler, and M. Luniak. 2006. Sequence analysis with Stata. *Stata Journal* 6: 435–460.

Cornwell, B. 2015. *Social Sequence Analysis: Methods and Applications*. New York: Cambridge University Press.

Elzinga, C. H. 2007. Sequence analysis: Metric representations of categorical time series. Technical report, Department of Social Science Research Methods, Vrije Universiteit, Amsterdam.

Elzinga, C. H., and A. C. Liefbroer. 2007. De-standardization of family-life trajectories of young adults: A cross-national comparison using sequence analysis. *European Journal of Population* 23: 225–250.

Forestier, G., F. Lalys, L. Riffaud, B. Trelhu, and P. Jannin. 2012. Classification of surgical processes using dynamic time warping. *Journal of Biomedical Informatics* 45: 255–264.

Gabadinho, A., G. Ritschard, M. Studer, and N. S. Müller. 2009. Mining sequence data in R with the TraMineR package: A user's guide for version 1.2. Technical report, Department of Econometrics and Laboratory of Demography, University of Geneva, Switzerland.

Halpin, B. 2010. Optimal matching analysis and life-course data: The importance of duration. *Sociological Methods and Research* 38: 365–388.

———. 2012. Sequence analysis of life-course data: A comparison of distance measures. Working Paper WP2012-02, Department of Sociology, University of Limerick. http://www.ul.ie/sociology/pubs/wp2012-02.pdf.

———. 2013. Sequence analysis. In *Oxford Bibliographies in Sociology*, ed. J. Baxter. New York: Oxford University Press.

———. 2014. Three narratives of sequence analysis. In *Advances in Sequence Analysis: Theory, Method, Applications*, ed. P. Blanchard, F. Bühlmann, and J.-A. Gauthier, 75–103. Berlin: Springer.

———. 2016a. Cluster analysis stopping rules in Stata. Working Paper WP2016-01, Department of Sociology, University of Limerick. https://osf.io/rjqe3.

———. 2016b. Multiple imputation for categorical time series. *Stata Journal* 16: 590–612.

Halpin, B., and T. W. Chan. 1998. Class careers as sequences: An optimal matching analysis of work-life histories. *European Sociological Review* 14: 111–130.

Hollister, M. 2009. Is optimal matching suboptimal? *Sociological Methods and Research* 38: 235–264.

Hubert, L., and P. Arabie. 1985. Comparing partitions. *Journal of Classification* 2: 193–218.

Jann, B. 2005. moremata: Stata module (Mata) to provide various functions. Statistical Software Components S455001, Department of Economics, Boston College. https://ideas.repec.org/c/boc/bocode/s455001.html.

Lesnard, L. 2008. Off-scheduling within dual-earner couples: An unequal and negative externality for family time. *American Journal of Sociology* 114: 447–490.

Marteau, P.-F. 2008. Time warp edit distance. ArXiv Working Paper No. arXiv:0802.3522. https://arxiv.org/abs/0802.3522.

———. 2009. Time warp edit distance with stiffness adjustment for time series matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31: 306–318.

Martin, P., I. Schoon, and A. Ross. 2008. Beyond transitions: Applying optimal matching analysis to life course research. *International Journal of Social Research Methodology* 11: 179–199.

McVicar, D., and M. Anyadike-Danes. 2002. Predicting successful and unsuccessful transitions from school to work by using sequence methods. *Journal of the Royal Statistical Society, Series A* 165: 317–334.

Raab, M., A. E. Fasang, A. Karhula, and J. Erola. 2014. Sibling similarity in family formation. *Demography* 51: 2127–2154.

Reilly, C., C. Wang, and M. Rutherford. 2005. A rapid method for the comparison of cluster analyses. *Statistica Sinica* 15: 19–33.

Studer, M., and G. Ritschard. 2016. What matters in differences between life trajectories: A comparative review of sequence dissimilarity measures. *Journal of the Royal Statistical Society, Series A* 179: 481–511.

Studer, M., G. Ritschard, A. Gabadinho, and N. S. Müller. 2011. Discrepancy analysis of state sequences. *Sociological Methods and Research* 40: 471–510.

Vinh, N. X., J. Epps, and J. Bailey. 2009. Information theoretic measures for clusterings comparison: Is a correction for chance necessary? In *Proceedings of the Twenty-Sixth International Conference on Machine Learning*, ed. L. Bottou and M. Littman, 1073–1080. Montreal, Canada: IMLS.

About the author

Brendan Halpin is a senior lecturer and the head of the Department of Sociology at the University of Limerick in Ireland and has a longstanding interest in longitudinal social science data.

A SADI components

This section gives fuller details on the various `SADI` commands. For even more detail, see the help entries for the commands once you install them.

A.1 Distance measures

`combinadd` *varlist*`,` `nspells(`*#*`)` `nstates(`*#*`)` `pwsim(`*string*`)` [`rtype(`*string*`)` `workspace` `maxtuples(`*#*`)`]

`combinadd` calculates a version of Elzinga's duration-weighted number of common subsequences measure for spell-structured data. The dataset needs to be restructured using `combinprep` before invoking `combinadd`.

`dynhamming` *varlist*`,` `pwdist(`*matname*`)` [`ref(`*#*`)` `mawindow(`*#*[*#*[*#*]]`)`]

`dynhamming` calculates Lesnard's dynamic Hamming distances between all pairs of sequences in the data, where *varlist* is a consecutive set of variables describing the elements of the sequence. Dynamic Hamming distances compare sequences element by element, such that the intersequence distance is the sum of the elementwise distances. The elementwise distances are dynamic based on the time-dependent structure of transition rates.

`sdhamming` *varlist*`, subsmat(`*matname*`) pwdist(`*matname*`)`

`sdhamming` calculates Hamming distances between all pairs of sequences in the data. Hamming distances compare sequences element by element, such that the intersequence distance is the sum of the elementwise distances.

`sdhollister` *varlist*`, subsmat(`*matname*`) timecost(`#`) localcost(`#`)`
`length(`*var*`) pwdist(`*matname*`) [ workspace standard(`*string*`) ]`

`sdhollister` calculates localized optimal matching distances between all pairs of sequences in the data, where *varlist* is a consecutive set of variables describing the elements of the sequence. It uses a Stata plugin implementation of Mattissa Hollister's adaptation of the Needleman–Wunsch algorithm.

`oma` *varlist*`, subsmat(`*matname*`) indel(`#`) length(`*var*`) pwdist(`*matname*`)`
`[ ref(`#`) workspace dups standard(`*string*`) ]`

`oma` calculates optimal matching distances between all pairs of sequences in the data, where *varlist* is a consecutive set of variables describing the elements of the sequence. It uses a Stata plugin implementation of the Needleman–Wunsch algorithm.

`omav` *varlist*`, subsmat(`*matname*`) indel(`#`) length(`*var*`) pwdist(`*matname*`)`
`[ facexp(`*real*`) workspace dups standard(`*string*`) ]`

`omav` calculates duration-adjusted optimal matching distances between all pairs of sequences in the data, where *varlist* is a consecutive set of variables describing the elements of the sequence. It uses a Stata plugin implementation of an adapted Needleman–Wunsch algorithm. It differs from the standard `oma` command in that the costs of elementary operations are reduced for tokens that are elements of runs of the same value. By default, the cost of an operation on an element of an n-element sequence is changed by a factor of $1/n^f$, where f is given by the `facexp()` option. The default is `facexp(0.5)`. A value of f of 0 produces the same result as `oma`, and a value of f of 1.0 weights all spells the same regardless of length.

Note: this measure is not guaranteed to be metric.

`twed` *varlist*`, subsmat(`*matname*`) lambda(`#`) nu(`#`) length(`*var*`)`
`pwdist(`*matname*`) [ ref(`#`) workspace dups standard(`*string*`) ]`

`twed` calculates Marteau's time-warp edit distance between all pairs of sequences in the data. Time-warping stretches and compresses the time dimension to achieve alignment in a manner similar but not identical to `oma`'s insertion and deletion. Because it uses compression instead of deletion, it respects the spell structure of the trajectory more than `oma` does.

A.2 Graphical summaries

`sdchronogram` *varlist* [*if*] [*in*] [, `by(`*string*`)` `textsize(`*string*`)` `proportional` *twoway_options*]

`sdchronogram` takes a set of sequences described by *varlist* in wide format and graphs the time-dependent distribution of the state variable. This is sometimes called a chronogram, or the transversal state distribution.

`trprgr` *varlist* [, `floor(`*real*`)` `ceiling(`*real*`)` `gmax(`#`)` `movingaverage(`#`)` `textsize(`*string*`)`]

`trprgr` takes a set of sequences described by *varlist* in wide format and graphs the time-dependent transition rate structure. The graphic consists of m rows and $m+1$ columns, where m is the number of states. The first column displays the time-dependent distribution of states, and the remaining $m \times m$ structure reproduces an $m \times m$ transition table but with graphs of time series of transition rates instead of single values.

A.3 Cluster related tools

`ari` *var1 var2* [*if*] [*in*]

`ari` calculates the adjusted Rand index comparing two "unlabeled" classifications (for example, two cluster solutions). It indexes agreement between the two classifications by counting pairs: the more that pairs that are in the same category in one classification are in the same category in the other classification (and different in different), then the higher the adjusted Rand index.

`permtab` *rowvar colvar* [*if*] [*in*] [, `gen(`*newvar*`)` `algorithm(`*string*`)` `random tables` `maxiter(`*real*`)`]

`permtab` permutes the columns of the cross-tabulation of *rowvar* by *colvar* to maximize Cohen's κ. It is intended for use in comparing cluster solutions where the identity of categories from one solution to the other is only defined in terms of membership. κ measures the excess of observed over expected on the diagonal. $\kappa_{\max}$ is the κ of the best solution and is reported. This is slow for more than 8 categories and becomes impossible above about 11.

A.4 Summary variables

`cumuldur` *varlist*`, cdstub(`*string*`) nstates(`#`)`

`cumuldur` creates variables holding the cumulative duration in each state in a sequence.

`sdentropy` *varlist*`, generate(`*string*`) cdstub(`*string*`) nstates(`#`)`

`sdentropy` creates a new variable holding the Shannon entropy of the sequence.

`nspells` *varlist*`, generate(`*string*`)`

`nspells` creates a variable holding the number of spells in a sequence.

`sdstripe` *varlist*`, generate(`*newvar*`)` [`symbols(`*string*`) xt xtspellsep(`*string*`) xtdursep(`*string*`)`]

`sdstripe` creates a single string variable representing a sequence, in either long format (each element of the sequence represented by a symbol) or spell format (each spell represented by a symbol indicating its state and a number indicating its duration).

A.5 Distance matrix tools

`corrsqm` *matrix1 matrix2* [*if*] [*in*] [`, nodiag`]

`corrsqm` takes two symmetric matrices of the same dimension (for example, distance matrices) and returns their correlation. More specifically, it returns the correlation between their lower triangles, optionally excluding the diagonal.

`sddiscrep` *groupvar*`, distmat(`*string*`) idvar(`*varname*`)` [`niter(`#`) dcg(`*string*`)`]

`sddiscrep` calculates Studer et al.'s (2011) measure of the discrepancy of a distance matrix, grouped by a categorical variable *groupvar*. The pseudo-R^2 and pseudo-F statistic are based on the extent to which the average distance to the centers of the groups are less than the average distance to the center of the ungrouped distance matrix. The p-value is based on permutations (100 by default, but Studer et al. [2011] recommend 1,000 to 5,000; set it to 1 for speed if you are not interested in the p-value). It optionally saves the casewise distance to the center of the cluster as a new variable.

`metricp` *matname* [`, countlimit(`#`) detailed`]

`metricp` takes a matrix of pairwise distances and tests that the triangle inequality is observed. If it finds triads infringing on the inequality, it reports at most 10 before

stopping (this is changed with the `countlimit()` option; set that to 0 for no limit). If there are no infringing cases and the matrix is large, it can be a little slow (tens of seconds). It is even slower with the `detailed` option (minutes), which identifies the infringing trio of sequences; without this option, only the fact that there is a shorter route between sequence i and sequence j is reported.

A.6 Helper utilities

`combinprep, state(`*string*`) length(`*string*`) idvar(`*varname*`) nspells(`*varname*`)`

`combinprep` takes sequence data in wide calendar format (that is, a consecutive string of numbered state variables representing state in each time unit, with one case per sequence) and turns it into wide spell format (consecutive pairs of numbered state and duration variables) with a separate variable indicating the number of spells. It returns the maximum number of spells observed in `r(maxspells)` and the range of the state variable in `r(nels)`. This can be used to prepare the data for `combinadd` and other techniques that focus on spell history rather than state history.

`maketrpr` *varlist*`, matrix(`*matname*`)` [`ma(#)`]

`maketrpr` takes a set of sequences described by *varlist* in wide format and creates an $n \times (n \times t)$ matrix where each $n \times n$ section contains the smoothed transition rates for the corresponding time period. It uses `tssmooth` to create the smoothed rates, defaulting to a three-unit look-head and look-back (that is, a seven-wide moving average). If the number of states is 4 and there are 10 periods, it generates a $(4 \times (10 - 1)) \times 4$ or 36×4 matrix, where $T[1..4, 1..4]$ contains the transition rates for time 1–2, $T[5..8, 1..4]$ for time 2–3, and so on.

This is essentially a utility program and is used by `dynhamming` and `trprgr`.

`trans2subs` *state* [*if*] [*in*]`, idvar(`*id*`) subsmat(`*matname*`)` [`diagincl`]

`trans2subs` calculates a substitution matrix based on observed transitions in the state variable and puts it in the subsmat matrix. The data must be in long format, sorted by `idvar()` and time.

B Compiling plugins

The C code for distance measures is in two main files, `omamatv3.c` and `elzspelladd.c`, available as ancillary files with the package. These must be compiled with `uthash.h` by Troy D. Hanson (http://troydhanson.github.io/uthash/index.html), which provides hash functions used in `elzspelladd.c`, and with `stplugin.c` and `stplugin.h`. The file `uthash.h` is provided as an ancillary. Both `stplugin.c` and `stplugin.h` are provided by Stata. For more information, see StataCorp's instructions for compiling plugins at http://www.stata.com/plugins/.

You may find updated versions of `stplugin.c` and `stplugin.h` at the Stata site and of `uthash.h` at http://troydhanson.github.io/uthash/index.html, but these are the versions used to create the published `SADI` plugins.

The published plugins are compiled for Windows and Linux, 32- and 64-bit, by cross-compilation on a 64-bit Linux system and for Mac OS by native compilation.