ORION*

A Software Package for Analysis of Resource Utilization Evolution

Version 1.0.0

David C. Anastasiu and George Karypis

Department of Computer Science & Engineering
University of Minnesota
Minneapolis, MN 55455

Contact email: dragos@cs.umn.com

March 14, 2016

*ORION is copyrighted by the regents of the University of Minnesota. Related papers are available via WWW at URL: http://www.cs.umn.edu/~dragos

Contents

1 Introduction

Understanding how utilization of resources evolves over time is an important task with diverse business applications. For example, an analysis of how PC usage evolves over time can help provide the best overall user experience for current customers, can help determine when they need brand new systems vs. upgraded components, and can inform future product design to better anticipate user needs.

As a way to understand usage evolution, consider application (resource) usage in a computer or mobile device. The variables being observed may describe daily time spent by a user interacting with a given application. Figure 1 illustrates the idea of computer usage and its evolution at a high level, grouping individual applications used by users into categories, such as Productivity and Games, and representing usage in these categories as vectors. A user's behavioral pattern may change over time. For example, our hypothetical user has a decreased overall PC usage as time progresses. Towards the end of the sequence, she spends more time surfing the Web, and less time using productivity tools. Characterizing usage evolution of many users enables us to find groups of users with similar trends (right-hand side of Figure 1), which may also benefit computer hardware and software designers by providing design feedback and insight into upcoming trends.

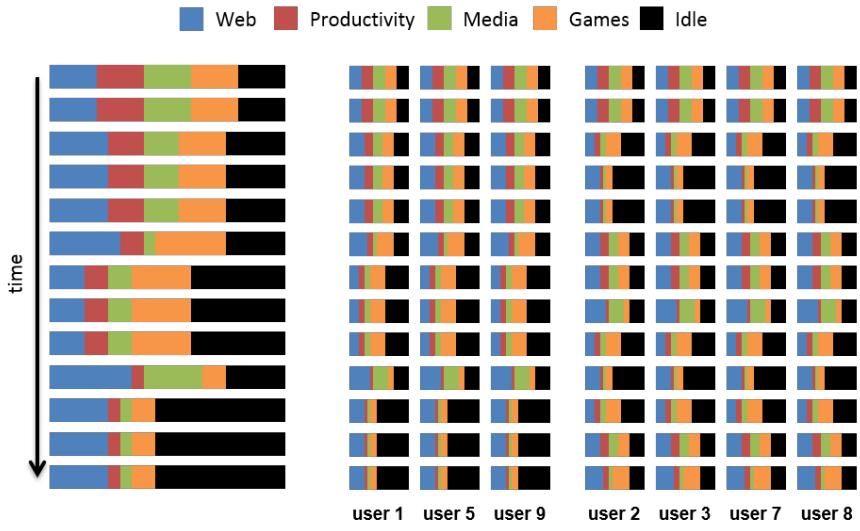

Figure 1: Computer usage evolution: a user's sequence of PC usage vectors (on the left), and sequences of two similar sets of users (on the right). (Best viewed in color.)

2 Overview of Orion

Orion is a serial software package that facilitates evolution analysis of multivariate resource utilization time series data, such as PC usage. Orion has been developed at the Department of Computer Science & Engineering at the University of Minnesota and is freely distributed. Its source code can be downloaded directly from *http://www.cs.umn.edu/~dragos/orion*.

Orion aims to characterize how a set of users utilize resources over time by finding prototypical usage patterns (*protos*) shared by users at different times within their usage history. To do so, it models each user's usage evolution as a sequence of protos and then performs a cross-user usage segmentation, i.e., a segmentation of the sequences of all users such that the error associated with modeling each segment by one of the protos is minimized. The multivariate time series segmentation problem has been previously addressed in the data mining community (see [1,2,4,5]), yet its goal has been the optimal segmentation of a single time-series. Instead, Orion finds the optimal segmentation of many time-series by a set of previously unknown building blocks, which it learns during the search. The details of the Orion algorithm can be found in [3].

3 The Orion program

ORION provides a stand-alone program that can be used to analyze multivariate time-series resource utilization data. Additionally, several scripts enable visualization of ORION's analysis output data. We describe first the format of input data required by ORION, then ORION's execution, output, and visualization scripts.

3.1 Input file formats

ORION expects as input *sparse* multivariate time-series data. Optionally, a file containing names/labels for each of the features in the time-series data can be provided. The formats of these input files are described in the following sections.

3.1.1 Time series data

The primary input for ORION is sparse resource utilization time-series data for a set of users. Each user's time-series is a sequence of sparse observation vectors, each of which represents the user's multivariate resource utilization in a time period (e.g., a week). Let the observation vectors of a user be stored as the rows of a matrix, ordered in increasing time order. ORION's input consists of a *sparse matrix file* containing vertically concatenated matrices for each of the user observation sequences, and a *counts file*, which specifies the number of rows in the matrix file for each user sequence.

There may be gaps in a user's time series, as they may not be using resources in every time period. These empty observation vectors should not be included in their sequence. Additionally, a user's first observation vector may not be in the same real time period as that of another user. ORION analyzes relative usage changes/evolution. As such, while the order of vectors in each user's time-series is important, the exact time periods when the observations were made are not.

ORION accepts several ASCII and binary sparse matrix input file types. The format of the input matrix file will be determined from the file's extension. In the event the matrix format cannot be determined from the extension, ORION will assume the input file is either in CSR or Cluto formats. Datasets stored in binary formats usually take up less space and load faster during execution. All binary formats assume little-endian encoding. The following describes each of the acceptable sparse matrix file formats.

- The *CSR* (`.csr` extension) and *Cluto* (`.clu`) formats represent a sparse matrix row-wise in ASCII files, as `<column-id, value>` pairs. Only the non-zero entries of the matrix are stored. Column ID numbering starts at 1. The Cluto format contains an additional header row with metadata information, three integers separated by space: the number of rows, the number of columns, and the number of non-zero values (nnz).

- The *Triplet CSR* (`.ijv`) format stores a sparse matrix in an ASCII file, containing a line for each value in the matrix, in the format "%d %d %f\n" (row id, column id, value).

- The *Binary Row-wise CSR* (`.binr`) format stores two 4-byte integers (number of rows and number of non-zero values), followed by 3 arrays. Let n be the number of rows in the matrix and nnz its number of non-zero values. The first array is a 4-byte integer pointer array (ptr) of length $n + 1$ containing pointers into the next two arrays, the indicators (ind) and values (val) arrays. These pointers specify where each row starts, s.t. row i's values are stored in the val array starting at index $ptr[i]$ and ending at index $ptr[i + 1]$. The indicators array ind stores column IDs associated with values at the same index in the val array. Needless to say, $ptr[0] = 0$ and $ptr[n] = nnz$. The ind array is a 4-byte integer array of length nnz, and the val array is a 4-byte float array of length nnz.

- The *Binary Triplet CSR* (`.bijv`) format stores four 4-byte integers (number of rows, number of columns, nnz, and $writevals$). The variable $writevals$ is 1 if values are included and 0 otherwise. If values exist, the file then contains nnz (row id, column id, value) triplets written as binary (int, int, float). Otherwise, it contains nnz (row id, column id) pairs written as binary (int, int).

The *counts file* is an ASCII file, with as many lines as the number of users/sequences, each line containing the observation count (sequence length) of the associated user sequence, i.e., the number of row in the

observation sequence matrix, following the same user order as the input matrix. By default, the counts file is assumed named <fstem>.counts, where <fstem> is the name of the input matrix file. The parameter *-countsfile* allows specifying an alternate name.

3.1.2 Feature labels

Some of the analysis results that ORION provides makes reference to features in the data (e.g. applications whose utilization was observed) by name/label. If available, feature labels can be provided in an ASCII file, one per line for each feature in the data, in the order corresponding to the feature/column IDs in the input matrix. By default, the labels file is assumed named <fstem>.clabels, where <fstem> is the name of the input matrix file. The parameter *-clabelsfile* allows specifying an alternate name for the feature label file. In the event that the *clabelsfile* file cannot be found, labels will be automatically generated, in the format "*l*<fid>", where <fid> is the feature id.

3.2 Using Orion

ORION is run via the orion executable. We start by listing its runtime options, and then describe the analysis information it outputs.

orion [options] fstem nprotos

Description

Orion performs multivariate resource utilization time series evolution analysis.

Parameters

fstem The name of the file that contains the multivariate resource utilization time series data (Section 3.1.1).

nprotos The number of protos that should be used to encode the utilization sequences.

Options

-rowmodel=text Specifies how the values will be scaled in each row.
Possible values are:
- none No scaling.
- log Take the log of the raw values. [default]
- sqrt Take the square-root of the raw values.

-colmodel=text Specifies how the values will be scaled in each column.
Possible values are:
- none No scaling. [default]
- idf Scale by the inverse document frequency across all usage vectors.
- sidf Scale by the inverse document frequency across sequences.

-minspan=int Specifies the minimum number of consecutive weeks that a proto must cover in a user's time-series.
Default value is 5.

-ncliters=int Specifies the maximum number of initial clustering iterations.
Default value is 20.

-niters=int Specifies the maximum number of segmentation refinement iterations.
Default value is 20.

-scost=float A per segment cost as a fraction of the error.
Default value is .01.

-mintp=float Specifies the minimum proto-to-proto transition probability to be analyzed.
Default value is .20.

-n2frac=float Specifies the maximum fraction of a vector to be analyzed for features.
Default value is .80.

-countsfile=text Required input file containing the number of rows in <fstem> for each sequence.
Default value is <fstem>.counts.

-clabelsfile=text Optional input file for the feature labels.
Default value is <fstem>.clabels. (labels will be generated if file does not exit).

-writepaths Outputs the proto paths of each sequence.

-pathsfile=text Output file for the proto paths of each sequence.
Default value is <fstem>.paths.c<scost>.s<minspan>.p<nprotos>.

-writeftrs Outputs the key features for each proto.

-ftrsfile=text Output file for the key features for each proto.
Default value is <fstem>.ftrs.c<scost>.s<minspan>.p<nprotos>.

-writep2pt Outputs the global proto-to-proto transition probabilities.

-p2ptfile=text Output file for the global proto-to-proto transition probabilities.
Default value is <fstem>.p2pt.c<scost>.s<minspan>.p<nprotos>.

-writetrans Outputs transition probabilities for a given transition level.

-transfile=text Output file for the transition probabilities for a given transition level.
Default value is <fstem>.trans.c<scost>.s<minspan>.p<nprotos>.l<translevel>.

-translevel=int Transition level to output transitions for. Must be > 0.
Default value is 1.

-writetinfo Outputs frequent transitions and their key discriminative features.

-tinfofile=text Output file for frequent transitions and their key discriminative features.
Default value is <fstem>.tinfo.c<scost>.s<minspan>.p<nprotos>.

-writeprotos Outputs the prototypical usage vectors.

-protosfile=text Output file for the prototypical usage vectors.
Default value is <fstem>.protos.c<scost>.s<minspan>.p<nprotos>.

-seed=int Specifies the seed for clustering.
Default value is 1.

-dbglvl=int Specifies the level of progress/debugging information to be displayed.
Possible values are (2 and above can be combined in binary form):

- 0 Disable all standard output.
- 1 Print general progress and output information. [default]
- 2 Include the protos.
- 4 Include per-sequence errors.
- 8 Include dynamic programming algorithm progress information.

-help Displays the command-line options along with a description.

With default *-dbglvl* parameters, `orion` prints all analysis information to `stdout` and does not write any files. The *-write** and their associated *-*file* parameters can be used to store analysis details in files. In the following, we describe `orion` program output under the default debug level, *-dbglvl=1*. Section 3.3 describes possible output files for `orion`.

```
orion -niters 3 -ncliters 5 seq1.csr 5
Reading matrix seq1.csr...
Reading user sequence counts...
Generating feature labels...

-----------------
#seqs: 100, #vecs: 10010, #dims: 545, #protos: 5
rowmodel: log, colmodel: none, minspan: 5, niters: 3
scost: 0.010, mintp: 0.200, n2frac: 0.800

Initial clustering:
  iter:  0, totsqe: 1.4513e+03, nconsecutive:   5510/ 10010
  iter:  1, totsqe: 9.0365e+02, nconsecutive:   4883/ 10010
  iter:  2, totsqe: 8.8820e+02, nconsecutive:   4699/ 10010
  iter:  3, totsqe: 8.7484e+02, nconsecutive:   4598/ 10010
  iter:  4, totsqe: 8.7323e+02, nconsecutive:   4598/ 10010
```

Figure 2: Initial clustering output in `orion`

3.2.1 Execution information

Figure 2 shows the beginning of the output produced by `orion`. After displaying runtime parameters and some statistics on the input data (*#seqs* - number of sequences/users, *#vecs* - number of observation vectors across all sequences, *#dims* - number of features in the observation vectors), `orion` displays execution statistics for the k-means clustering iterations used to identify the initial protos. After each iteration, *totsqe* shows the sum of the square root error between the observation vectors and their respective protos (the centers of the clusters they are assigned to), and *nconsecutive* shows the number of observation vectors consecutively assigned to the same proto in the user sequences.

Execution in `orion` continues by displaying the proto-to-proto distance matrix (see Section 3.2.2) and the proto-to-proto transition matrix (see Section 3.2.3) for the initial protos. In the second phase of the program, after encoding usage sequences with the identified protos, `orion` employs an iterative process by which the protos are automatically derived from the segmentation and an optimal segmentation is determined from the protos using a dynamic programming algorithm. After each iteration, the program displays the latest computed objective function value. Additional details about the algorithms implemented in `orion` can be found in [3]. After the final protos and segmentations are computed, `orion` displays analysis results, as described in Sections 3.2.2–3.2.5.

3.2.2 Proto distances

```
Proto distances -------------
 189.8 =>     0.0  400.7  179.3  295.1  297.3
 292.9 =>   400.7    0.0  298.6  410.1  432.8
  49.1 =>   179.3  298.6    0.0  177.4  179.1
 187.4 =>   295.1  410.1  177.4    0.0  295.8
 189.9 =>   297.3  432.8  179.1  295.8    0.0
End proto distances --------------
```

Figure 3: Proto distances matrix output in `orion`

The proto distances output includes, for each proto, the proto intensity score, followed by squared euclidean distances from the proto to each of the protos, in ascending order of proto ID. Figure 3 exemplifies output of a proto distance matrix for 5 protos.

A proto is, in essence, the centroid of the set of resource utilization vectors encoded by the proto after

segmentation across all users, and is thus not normalized. The proto intensity score is the squared ℓ^2-norm of the proto and measures the magnitude of the resource utilization values represented by the proto.

Proto distance is an indicator of how similar two protos are. A large number of small proto-to-proto distance values may be an indication that the requested number of protos is too high.

3.2.3 Proto-to-proto transition matrix

```
Proto-to-proto transition matrix -------------
   19 =>  0.000 0.050 0.900 0.000 0.000
    9 =>  0.300 0.000 0.400 0.100 0.100
   19 =>  0.500 0.250 0.000 0.150 0.050
    2 =>  0.000 0.000 0.333 0.000 0.333
    3 =>  0.000 0.000 0.250 0.500 0.000
End proto-to-proto transition matrix -------------
```

Figure 4: Proto-to-proto transition matrix output in `orion`

The proto-to-proto transition matrix includes global transition probability values, computed over all transitions occurring in the proto sequences of all users. Figure 4 exemplifies output of a proto-to-proto transition matrix for 5 protos. For each proto, `orion` outputs the number of transitions from the proto to any other proto, followed by the probability that usage transitions from that proto to each of the protos, sorted in increasing proto ID order.

3.2.4 Proto features

```
Proto features -----
Proto: 0 [814 172.31 92.78]
  0.061 0.066 +0.024 0.061 1403
  0.046 0.060 +0.014 0.107 1185
  0.041 0.058 +0.011 0.149 176
  0.039 0.005 +0.049 0.188 1363
...
Proto: 4 [54 147.79 77.96]
  0.222 0.065 +0.205 0.222 1294
  0.049 0.004 +0.065 0.271 1313
  0.042 0.004 +0.055 0.314 1347
...
  0.012 0.004 +0.011 0.806 1254
End proto features-----
```

Figure 5: Key proto features output in `orion`

For each of the identified protos, `orion` shows analysis information for key features, in the following format,

```
Proto:  pid [pcount pint psqe]
  fint1 cint1 fdis1 sum1 flabel1
  fint2 cint2 fdis2 sum2 flabel2
  ... ,
```

which we further describe below. The list of features for each proto is sorted in decreasing order of descriptive power/feature intensity (`fint`). The parameter *-n2frac* determines the percent of features that are included in the output for each proto. Figure 5 gives an example of this type of key feature output. Let $\boldsymbol{p}$ be a proto,

and p_j the value/weight of the j-th feature within the proto. Furthermore, let $\boldsymbol{c}$ be the global *center*, the mean vector over all observation vectors in all user sequences, and c_j be similarly defined as p_j.

- `pid` is the integer Proto ID. Proto IDs start with 0.
- `pcount` shows the number of observation vectors that were encoded by the proto, across all user sequences.
- `pint` is the proto intensity score (see Section 3.2.2).
- `psqe` is the squared error between the proto and the center vector, $||\boldsymbol{p} - \boldsymbol{c}||_2^2$, and points to the discriminative power of the proto.
- `fint` is the feature intensity score, measuring the descriptive power of the feature within the proto, computed as $p_j^2/||\boldsymbol{p}||_2^2$.
- `cint` is the associated feature intensity score in the center vector, measuring the global descriptive power of the feature across all vectors, computed as $c_j^2/||\boldsymbol{c}||_2^2$.
- `fdis` is a measure of the discriminative power of the feature, computed as $(p_j - c_j)^2/||\boldsymbol{p} - \boldsymbol{c}||_2^2$. The added sign is an indicator whether $p_j > c_j$ $(+)$, or not, $(-)$.
- `sum` represents the cumulative feature intensity score for features displayed so far for the proto.
- `flabel` is the feature label.

3.2.5 Frequent proto-to-proto transitions

```
Frequent proto-to-proto transitions -------------
0 => 2 [18 0.900][172.31 44.67 110.74]
   -0.92 -2.509 0.038 0.001 0.05 l452
   -0.89 -2.225 0.039 0.002 0.10 l363
   -0.91 -2.443 0.034 0.001 0.14 l343
   -0.92 -2.501 0.033 0.001 0.18 l16
   -0.91 -2.364 0.033 0.001 0.23 l125
   -0.88 -2.133 0.034 0.002 0.27 l36
...
4 => 3 [2 0.500][147.79 171.60 85.01]
  +12.38 +2.555 0.000 0.065 0.11 l95
   +3.50 +1.491 0.004 0.074 0.20 l422
   -0.45 -0.597 0.222 0.058 0.28 l294
   +1.06 +0.717 0.033 0.121 0.34 l76
...
   -0.84 -1.833 0.014 0.000 0.80 l539
End frequent proto-to-proto transitions -------------
```

Figure 6: Frequent proto-to-proto transitions output in `orion`

The parameter *-mintp* specifies the minimum proto-to-proto transition probability to be analyzed. After computing probability values for all proto-to-proto transitions, ORION displays, in decreasing transition probability order, those proto transitions above the *-mintp* cutoff. For each transition, ORION displays some summary information about the protos involved in the transition, followed by key feature information related to the two protos. Figure 6 gives an example of this type of key proto transitions output. Let $\boldsymbol{p}$ and $\boldsymbol{q}$ be the two protos in a transition, s.t. usage transitions from $\boldsymbol{p}$ to $\boldsymbol{q}$, and let p_j and q_j be the value in the two protos for the j-th feature. The summary information is in the following format,

$\texttt{pid}_p$ `=>` $\texttt{pid}_q$ `[freq prob` $\texttt{int}_p$ $\texttt{int}_q$`, sqe]`,

which we further describe below.

- `pid`$_p$ and `pid`$_q$ are the proto IDs for the protos transitioning from and to.
- `freq` is the count of transitions from $\boldsymbol{p}$ to $\boldsymbol{q}$.
- `prob` is the probability of transitioning from $\boldsymbol{p}$ to $\boldsymbol{q}$.
- `int`$_p$ and `int`$_q$ are the intensity scores of the two protos (see Section 3.2.2).
- `sqe` denotes the squared Euclidean distance between the two protos.

For each selected proto transition, ORION displays a number of statistics for key features, in the format,
`rdist logr fint`$_p$ `fint`$_q$ `dsum flabel,`
which we further describe below. The first two numbers showcase the differences between the feature utilization in the two protos, while the next two show the respective feature magnitude in each of the two vectors. The list of features for each proto transition is sorted in decreasing order of feature discriminative power, $(p_j - q_j)^2/||\boldsymbol{p} - \boldsymbol{q}||_2^2$, where j is the ID of the chosen feature.

- `rdist` is the relative distance between the two feature values, computed as $|p_j - q_j|/p_j$. The added sign is an indicator whether $p_j \leq q_j$ $(+)$, or not, $(-)$.
- `logr` is the log ratio between the two feature values, $\log(q_j/p_j)$.
- `fint`$_p$ and `fint`$_q$ are the feature intensity scores in the respective protos (see Section 3.2.4).
- `dsum` is the cumulative discriminative power for features displayed so far for the proto transition.
- `flabel` is the feature label.

3.3 Output file formats

When invoked with certain parameters (see Section 3.2), ORION can store a number of analysis results, which are described in the following sections. All output files are ASCII text files.

3.3.1 Proto paths

ORION encodes user sequences as sequences of protos, which can be stored in the *pathsfile* file. The proto sequences are stored one per line, in the same order corresponding to the sequence matrices in the <`fstem`> input file. Each proto sequence is a list of proto IDs separated by space.

3.3.2 Proto key features

For each of the identified protos, the *ftrsfile* file stores key feature information for the proto, in the format,
`pid pcount pint psqe \t fint`$_1$ `cint`$_1$ `fdis`$_1$ `sum`$_1$ `flabel`$_1$ `fint`$_2$ `cint`$_2$ `fdis`$_2$ `sum`$_2$ `flabel`$_2$ ....
The format components are described in detail in Section 3.2.4.

3.3.3 Proto-to-proto transitions

The *p2ptfile* file stores the dense proto-to-proto global transition probability matrix, computed over all transitions occurring in the proto sequences of all users (see Section 3.2.3). For each source proto, in increasing proto id order, the file contains, on one line, the probabilities that usage transitions from the source proto to the another protos. Values in each line are separated by space and sorted in increasing proto ID order.

3.3.4 Level-wise proto transitions

Proto transitions can also be analyzed level-wise. Level k transitions are those transitions between the k-th proto and the $k+1$-th proto in a proto-sequence. The *transfile* file contains proto-to-proto transition probabilities, counting only transitions in the requested level, in sparse *Triplet CSR* format. Each line of the file contains three values, in the format, `pid`$_p$ `pid`$_q$ `prob`, where `pid`$_p$ and `pid`$_q$ are the proto IDs of the proto transitioning from and to, and `prob` is the probability of transitioning from $\boldsymbol{p}$ to $\boldsymbol{q}$. In addition, *start* (S) and *end* (E) probabilities are included. A start probability is stored as the triplet `S pid`$_p$ `prob`, and represents the probability of a sequence being encoded by proto $\boldsymbol{p}$ in level k. An end probability is stored as the triplet `pid`$_p$ `E prob`, and represents the probability of a sequence not transitioning to level $k+1$. Level-wise transition output can be graphically visualized using the `transitions.py` script (see Section 3.4.2).

3.3.5 Frequent proto-to-proto transitions

Frequent proto transitions and their key discriminative features are stored in the *tinfofile* file, in the format,
`pid`$_p$ `pid`$_q$ `freq prob int`$_p$ `int`$_q$ `sqe \t rdist logr fint`$_p$ `fint`$_q$ `dsum flabel rdist logr ...`.
For each included frequent transition, the initial 7 values describing the protos involved in the transition are separated by a TAB character from the list of sets of 6 feature characteristics of discriminating features for the transition, separated by spaces. The format components are described in detail in Section 3.2.5.

3.3.6 Proto vectors

The *protosfile* file stores the dense matrix whose columns are the protos, sorted in increasing proto ID order. Each line in the file represents a different feature and contains the proto values for the feature, separated by space. The feature label is also included at the end of the line.

3.4 Visualizing Orion output

Some of the `orion` output files can be used to create visualizations depicting proto transitions and/or evolutions. The following sections describe two Python scripts included in the ORION distribution for this purpose.

3.4.1 The sankey.py script

The `sankey.py` script creates a Sankey diagram depicting proto transitions up to a given number of levels. Figure 7 gives an example of a diagram created using the script. The script requires the *pathsfile* `orion` output file as input to create a proto evolution chart (see Section 3.3.1). Optionally, when invoked with the *--mode proto* or *--mode rproto* parameters, it can create proto Sankey evolution diagrams for each proto. In these modes, the script also requires the *ftrsfile* ORION output file (see Section 3.3.2), and uses the information therein to create pie charts displaying feature importance in each proto.

The output generated by the `sankey.py` script is an `html` file. The output Web page uses the JavaScript package `google.visualization.Sankey` from Google Charts[1] to create an interactive Sankey chart. When hovering over each transition edge in the chart, all others will be partially faded. Figure 8 shows usage information for the `sankey.py` script.

3.4.2 The transitions.py script

The `transitions.py` script depicts level-wise proto transitions stored in the *transfile* `orion` output file (see Section 3.3.4). Figure 9 gives an example of its output. The script requires the *pyplot* package and, by default, creates a figure that is interactively displayed. Optionally, the figure can be stored in a file compatible with *pyplot* file storage options (e.g., `png`, `eps`, `pdf`, etc.), and the interactive figure can pe disabled. Figure 10 shows usage information for the `transitions.py` script.

[1] https://developers.google.com/chart/

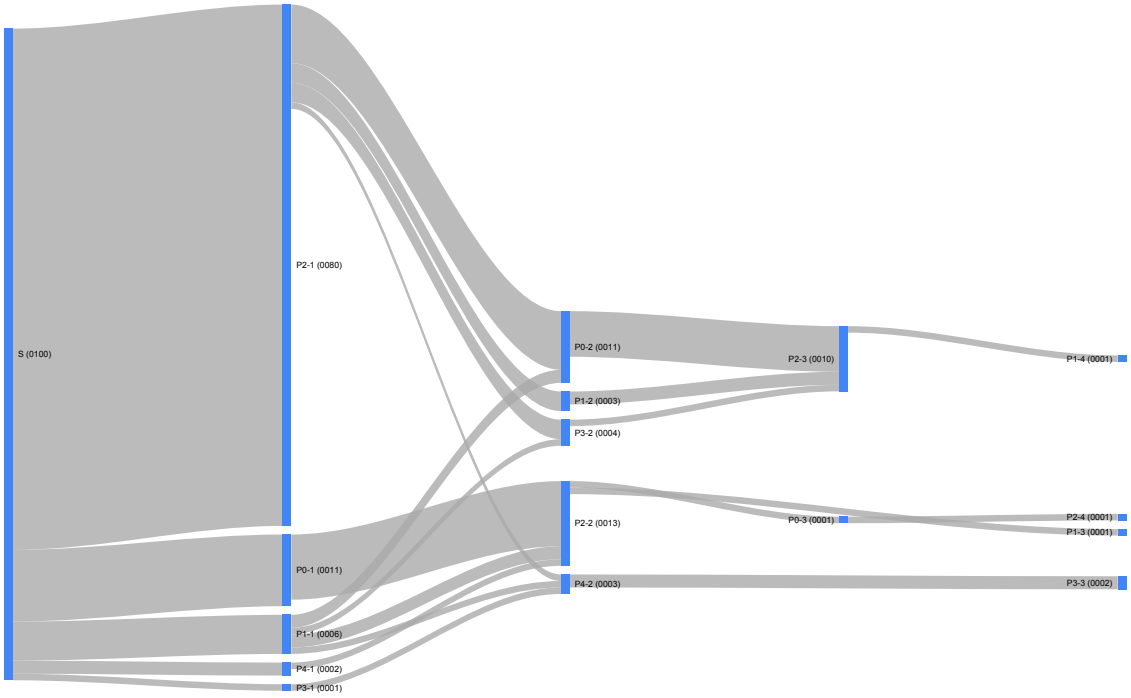

Figure 7: Proto evolution. Each (blue) vertical line represents a behavior state, either *Start* (S) or one of the p protos, and its extent is proportional to the proto level frequency. Curve (gray) lines show users transitioning from state to state. Proto states are labeled with their ID, level in sequence, and their frequency, i.e., the number of users that have that proto at that level in their proto sequences.

```
usage: sankey.py [-h] [-f MINFREQ] [-l MAXLEVEL] [-m MODE]
                 pathsfile [ftrsfile]

positional arguments:
  pathsfile             File containing the proto paths of each sequence.
  ftrsfile              File containing the key features for each proto
                             (required for modes proto and rproto).

optional arguments:
  -h, --help            show this help message and exit
  -f MINFREQ, --minfreq MINFREQ
                        Minimum frequency to include a transition in the chart.
  -l MAXLEVEL, --maxlevel MAXLEVEL
                        Maximum number of levels to show in the chart (for mode global).
  -m MODE, --mode MODE  Type of chart to show:
                            global  Show global evolution chart [default]
                            proto   Show per-proto evolution
                            rproto  Show reverse per-proto evolution
```

Figure 8: Usage information for the sankey.py script

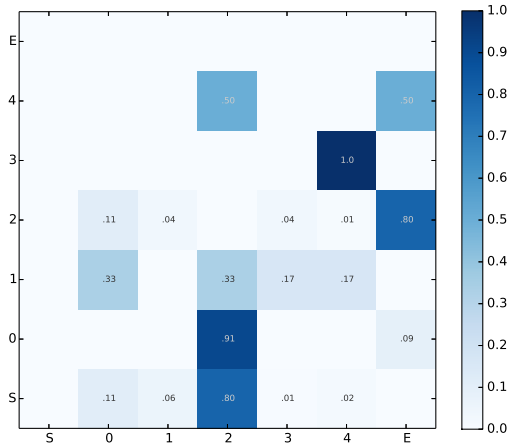

Figure 9: Proto level-1 transitions. *S* and *E* are the *Start* and *End* states, and the numbers denote protos. Each row shows the probabilities of a user transitioning from the proto identified by the row ID towards other protos, identified by column IDs. We only show probabilities above 0.01 in the transition matrix.

3.5 Execution examples

In this Section, we include some examples of usage evolution analysis with `orion` and its included visualization scripts. The distribution includes an example time series input file, in the `data` directory, named `seq1.csr`, and its associated counts file, `seq1.csr.counts`. An example feature labels file that does not conform to the default naming convention for a *clabelsfile* is also included. The included data are randomly generated based on real PC application usage data described in [3]. The examples below will use these data files.

- Execute an `orion` analysis, requesting 5 prototypical usage vectors, and print results to `stdout`.
  `~$ orion data/seq1.csr 5`
- Provide a custom feature labels file for the previous analysis.
  `~$ orion data/seq1.csr 5 -clabelsfile data/seq1.labels`
- Specify a smaller segment generation cost and fewer segmentation refinement iterations.
  `~$ orion data/seq1.csr 5 -scost 0.005 -niters 10`

```
usage: transitions.py [-h] [-ns] [-t TITLE] [-o OFILE] [-m MINP] transfile

positional arguments:
  transfile             File containing transition probabilities for a given
                        transition level.

optional arguments:
  -h, --help            show this help message and exit
  -ns, --noshow         Do not display the figure.
  -t TITLE, --title TITLE
                        Chart title.
  -o OFILE, --ofile OFILE
                        Output file for the chart (must have an accepted
                        pyplot output format extension).
  -m MINP, --minp MINP  Minimum transition probability to display text for.
```

Figure 10: Usage information for the transitions.py script

- Store resulting prototypical usage vectors using the default *protosfile* file name and suppress `stdout` output.
  `~$ orion data/seq1.csr 5 -writeprotos -dbglvl 0`
- Store level-1 proto transition probabilities in a custom file, and use it to generate a transition probability visualization chart.
  `~$ orion data/seq1.csr 5 -writetrans -transfile data/seq1.trans`
  `~$ scripts/transitions.py data/seq1.trans`
- Store the transitions chart in a `pdf` file, preventing the display of the interactive chart.
  `~$ scripts/transitions.py data/seq1.trans -o data/seq1.trans.pdf -ns`
- Execute an `orion` analysis, requesting 5 prototypical usage vectors, and store the *pathsfile* and *ftrsfile* output files using their default names.
  `~$ orion data/seq1.csr 5 -writepaths -writeftrs`
- Use the *pathsfile* to generate an usage evolution Sankey diagram, limited to the first 3 transition levels.
  `~$ scripts/sankey.py data/seq1.csr.paths.c0.010.s5.p5 -l 3 > data/sankey.html`
- Use the *pathsfile* and *ftrsfile* files to generate Sankey evolution diagrams for each proto.
  `~$ scripts/sankey.py data/seq1.csr.paths.c0.010.s5.p5 data/seq1.csr.ftrs.c0.010.s5.p5 -m proto > data/sankey2.html`

4 System requirements and contact information

ORION is written entirely in ANSI C, and is portable on most Unix systems that have an ANSI C compiler (the GNU C compiler will do). It has been tested on Linux and OSX. Instructions on how to build and install ORION can be found in the files `BUILD.txt` or `BUILD-Windows.txt` of the distribution.

ORION has been extensively tested on a number of different architectures. However, even though ORION contains no known bugs, this does not mean that all of its bugs have been found and fixed. If you have any problems, please send email to dragos@cs.umn.edu with a brief description of the problem. Also, any future updates to ORION will be made available on WWW at http://www.cs.umn.edu/~dragos/orion.

5 Copyright & license notice

ORION is copyrighted by the Regents of the University of Minnesota. It is licensed under the MIT License; you may not use this program except in compliance with the License. You may obtain a copy of the License

at: http://opensource.org/licenses/MIT.

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

References

[1] Janos Abonyi, Balazs Feil, Sandor Nemeth, and Peter Arva. Fuzzy clustering based segmentation of time-series. In Michael R. Berthold, Hans-Joachim Lenz, Elizabeth Bradley, Rudolf Kruse, and Christian Borgelt, editors, *Advances in Intelligent Data Analysis V*, volume 2810 of *Lecture Notes in Computer Science*, pages 275–285. Springer Berlin Heidelberg, 2003.

[2] Janos Abonyi, Balazs Feil, Sandor Nemeth, and Peter Arva. Modified gath–geva clustering for fuzzy segmentation of multivariate time-series. *Fuzzy Sets Syst.*, 149(1):39–56, January 2005.

[3] David C. Anastasiu, Al M. Rashid, Andrea Tagarelli, and George Karypis. Understanding computer usage evolution. In *31st IEEE International Conference on Data Engineering*, ICDE '15, 2015.

[4] Hongyue Guo, Xiaodong Liu, and Lixin Song. Dynamic programming approach for segmentation of multivariate time series. *Stochastic Environmental Research and Risk Assessment*, pages 1–9, 2014.

[5] N. Wang, X. Liu, and J. Yin. Improved gath-geva clustering for fuzzy segmentation of hydrometeorological time series. *Stochastic Environmental Research and Risk Assessment*, 26:139–155, 2012.