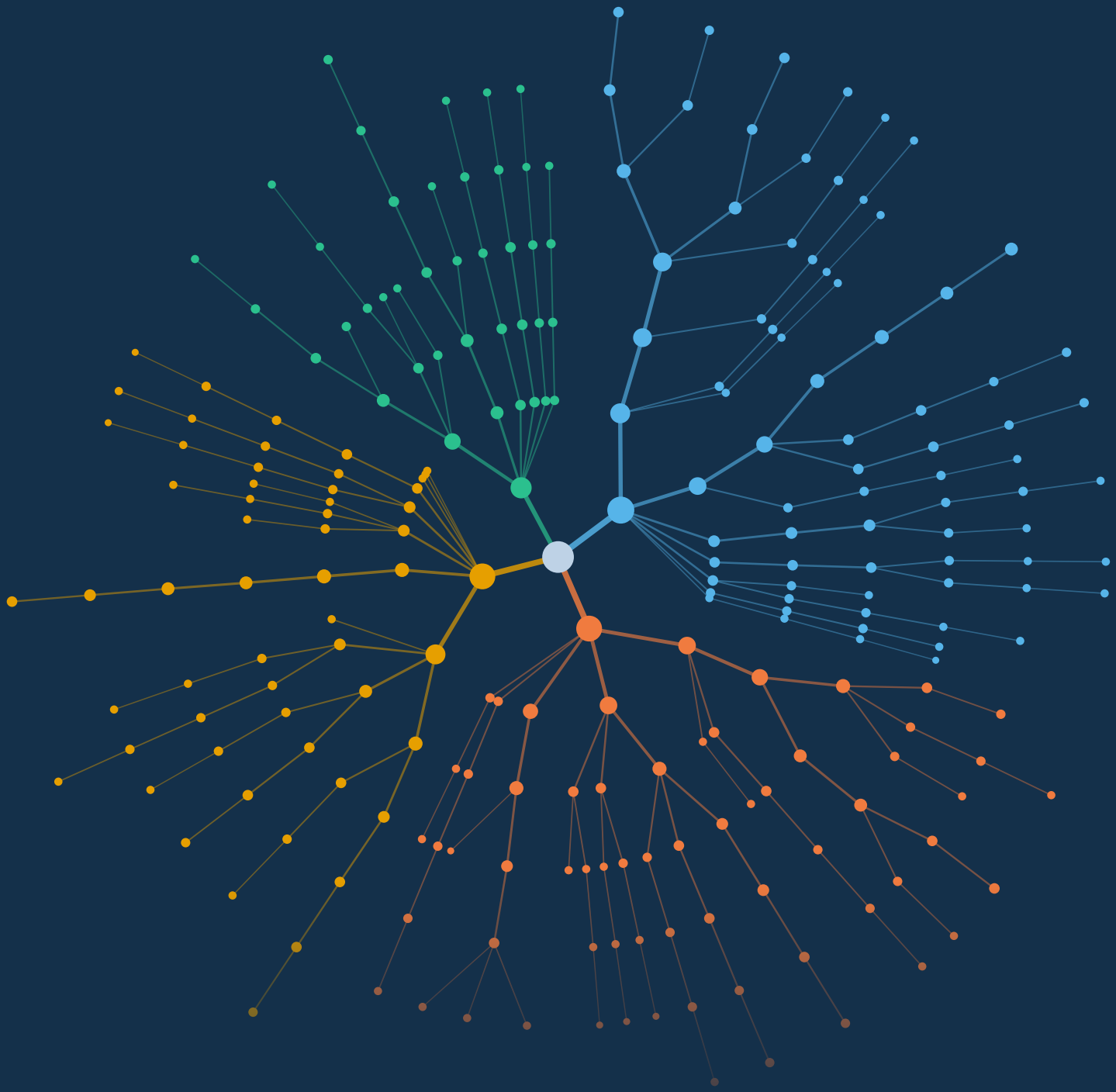


Mining and Visualizing Frequent Sequences in Python

The `spmviz` toolbox

János Abonyi · László Bántay
University of Pannonia, Veszprém





Mining and Visualizing Frequent Sequences in Python

The spmviz toolbox

János Abonyi
László Bántay
University of Pannonia, Veszprém

Version 1.0 · November 2026

Mining and Visualizing Frequent Sequences in Python. The `spmviz` toolbox

János Abonyi, László Bántay

Version 1.0, November 2026

© 2026 János Abonyi, László Bántay. This book is licensed under the Creative Commons Attribution 4.0 International License (CC BY 4.0, https://creativecommons.org/licenses/by/4.0/): it may be copied, redistributed, and adapted for any purpose, provided the source is credited. The `spmviz` software described in the book is released under the MIT License.

Software: https://github.com/ComplexSystemsIntelligence/spmviz
Documentation: https://complexsystemsintelligence.github.io/spmviz
Installation: `pip install spmviz`

The package extends the network-based visualization of frequent sequences of Bántay and Abonyi (PLOS ONE 19(5): e0301262, 2024). The bundled click-stream is an excerpt of the BPI Challenge 2016 data of the Dutch Employee Insurance Agency (UWV), the alarm log an anonymized log of an industrial alkylation plant published with the paper. The research summarized in this book was supported by the National Research, Development and Innovation Office of Hungary (NKFIH) through the OTKA 143482 project.

Typeset with LaTeX in Libertinus Serif, Raleway, and IBM Plex Mono. Every figure that shows a computed result was produced by an example script of the package. The cover shows the opening book of the chess example as a radial extension tree computed with `spmviz`: each ring is a ply, each branch a frequent move sequence, line widths proportional to the number of games.

Preface

Event logs are everywhere: the clicks of a web session, the alarms and operator actions of a chemical plant, the moves of a chess game, the activities of a working day. Sequential pattern mining condenses such logs into the sequences of events that many of them share. The condensation, however, comes at a price. A mining run easily returns hundreds or thousands of frequent sequences, most of them variations of each other, and a table of them tells little about the process that produced the data. The patterns have to be *seen*: which sequences grow out of which, where a process branches, which patterns belong to the same sub-process, how a group of users differs from another.

The origin of the book. In 2024 László Bántay and János Abonyi proposed three network-based views of frequent sequences [1]: a weighted network that links every sequence with its one-event extensions, and two maps obtained by multidimensional scaling, one from the confidences along the extension tree and one from the co-occurrence of the sequences in the same traces. The method grew out of the alarm-management research of the group [2, 3, 4], where operators have to understand alarm floods in minutes. The scripts of the paper [5] combined the SPMF library, pandas, NetworkX and pyvis. This book and its companion package, `spmviz`, turn them into a general toolbox and place the network-based views among the other visualizations of frequent sequences proposed in data mining, visual analytics, process mining and the social sciences.

What the package offers. `spmviz` contains five sequential pattern miners written in pure Python (PrefixSpan, SPAM, CM-SPAM, SPADE and GSP, all with gap, span and time constraints), closed, maximal and generator patterns, top-k mining, sequential rules and episodes, and adapters to the miners available in Python and Java. On top of the mined patterns it offers nearly forty static views, most of them with an interactive twin: the three network-based views of the paper and a fused fourth one, trees (sunburst, icicle, LifeFlow, word tree), flows (Sankey, directly-follows graph, MatrixWave, chord), matrices and maps (incidence, similarity, grouped rule matrix, measure scatter, lattice, UpSet), and the sequence views of social-science sequence analysis and process mining (index plot, chronogram, dotted chart, sequence logo). Several views are new: the continuation-entropy tree, the contrast sunburst, the growth Sankey, pattern-induced higher-order networks,

pattern landscapes, and community-based segmentation of the pattern set into sub-logs. Every graph is a NetworkX object, every table a pandas data frame, and every result can be written to SPMF, GraphML, GEXF, Cytoscape or JSON files, so that the analysis can continue in Gephi, Cytoscape, R or a spreadsheet.

Who the book is for. The book is written for data scientists, process engineers, web and business analysts, and graduate students who analyse event data. It assumes elementary probability and graph theory and some Python. Every method is stated in a box and summarized in a few bullets, and every chapter ends with examples that run on bundled data: two real logs of the original paper, small textbook databases, synthetic data whose structure is known, and the public benchmark databases of the sequential pattern mining literature.

How to run the examples. The package is installed with `pip install spmviz`. The scripts that produce the figures are in the `examples/` directory of the repository, grouped by chapter, and are listed in chapter C. The online documentation renders them as a gallery and as Jupyter notebooks. The code printed in the chapters is extracted from the same scripts, so it runs with the version of the package that the book describes.

Acknowledgments. The network-based views and the industrial alarm log come from the doctoral research of László Bántay at the University of Pannonia. We thank the authors of SPMF, NetworkX, pm4py and the other open-source libraries on which the package builds. The research summarized in this book was supported by the National Research, Development and Innovation Office of Hungary (NKFIH) through the OTKA 143482 project.

Veszprém, November 2026 The authors

Contents

Preface iii

1 Introduction 1

1.1 Problem and idea 1
1.2 What visualization can and cannot tell 2
1.3 The toolbox 3
1.4 A first session 4
1.5 How the book is organized 4
1.6 Further reading 5

2 Sequences, patterns and their mining 7

2.1 Problem and idea 7
2.2 Theory 8
2.3 Mining algorithms 10
2.4 The data model of spmviz 11
2.5 Using it in spmviz 12
2.6 Examples 13

3 A map of the visualizations of frequent sequences 19

3.1 Problem and idea 19
3.2 Five objects, five families 20
3.3 Which view answers which question 21
3.4 The catalogue 22
3.5 Further reading 24

4 Network-based visualization of frequent sequences 25

4.1 Problem and idea 25
4.2 Theory 26
4.3 Using it in spmviz 28
4.4 Reproducing the paper 29

4.5 Examples . . . 30

5 Trees of prefixes and patterns 39
5.1 Problem and idea . . . 39
5.2 Theory . . . 40
5.3 Using it in spmviz . . . 42
5.4 Examples . . . 43

6 Flows and transitions 51
6.1 Problem and idea . . . 51
6.2 Theory . . . 52
6.3 Using it in spmviz . . . 54
6.4 Examples . . . 54

7 Matrices, maps and sets 63
7.1 Problem and idea . . . 63
7.2 Theory . . . 64
7.3 Using it in spmviz . . . 66
7.4 Examples . . . 66

8 Sequence views and summaries 73
8.1 Problem and idea . . . 73
8.2 Theory . . . 74
8.3 Using it in spmviz . . . 77
8.4 Examples . . . 77

9 Interoperability 83
9.1 Problem and idea . . . 83
9.2 Data in . . . 83
9.3 Other miners . . . 85
9.4 Results out . . . 85
9.5 Interactive figures . . . 86

10 Benchmarks and case studies 89
10.1 Problem and idea . . . 89
10.2 Benchmark databases . . . 89
10.3 Run times . . . 90
10.4 Case study: an online benefit application . . . 92
10.5 Case study: early warning in alarm management . . . 94
10.6 Analysing a new log . . . 96

A API reference 99
A.1 `spmviz.data` 99
A.2 `spmviz.patterns` 100
A.3 `spmviz.mining` 100
A.4 `spmviz.backends` 100
A.5 `spmviz.measures` 101
A.6 `spmviz.networks` 101
A.7 `spmviz.embedding` 102
A.8 `spmviz.layout` 103
A.9 `spmviz.nbvfs` 103
A.10 `spmviz.summary` 103
A.11 `spmviz.predict` 103
A.12 `spmviz.viz` 104
A.13 `spmviz.interactive` 106
A.14 `spmviz.io` 106
A.15 `spmviz.datasets` 107

B From the NbVFS scripts to spmviz 109
Data preparation and mining 109
The three views 110
A complete translation 111

C The example gallery 113
C.1 First steps: data 113
C.2 Network-based visualization of frequent sequences 114
C.3 Trees: prefix trees 114
C.4 Flows: Sankey diagrams 114
C.5 Maps and matrices of pattern sets 115
C.6 Sequences: index plots 115
C.7 Applications and benchmarks 116

Bibliography 117

List of Figures

1.1 From an event log to an interpretable picture. 2

2.1 The database of the PrefixSpan paper and part of its extension tree. 9
2.2 A glyph table of the longest frequent sequences of the PrefixSpan database. 13
2.3 The pattern explosion: numbers of all, closed and maximal frequent sequences of the PrefixSpan database against the support threshold. 14
2.4 How three hundred days begin: the sunburst of the first four activities (rings are steps, angles numbers of days). 15
2.5 The frequent sequences of the daily routines as a pattern tree: edges as wide as the support, three most frequent continuations per node. 15
2.6 The ten most supported daily-routine patterns of length at least three. . . . 16
2.7 Run times of the five built-in miners on a thousand web-shop journeys; all return the same 890 patterns. 16
2.8 Sixty days as rows of colored activities, sorted by the highlighted patterns ⟨wake up coffee⟩ and ⟨commute work⟩, whose earliest occurrences are marked. 17

3.1 Five families of visualizations of event sequences and frequent sequences. 20

4.1 NBVFS-WN of the click-stream sessions starting with Hours worked: radial layout, node color = length, node size = cumulative confidence, edge width = confidence. 31
4.2 The extension trees of the first four services in the hierarchical layout, one row per length. 31
4.3 Pruning weak extensions of the Hours worked tree at minimum confidences 0, 0.2 and 0.4. 32
4.4 NBVFS-CM of the click-stream with three start events highlighted; edges join every sequence to its parent, node size is the support. 32
4.5 The CM map by classical scaling (left) and by SMACOF (right) with their stress values. 32
4.6 NBVFS-TM of synthetic alarm floods: the frequent sequences of the four interleaved sub-processes form four clusters. 33
4.7 The same map with the sequences of the cooling sub-process highlighted. 33

4.8 TM maps at minimum supports of 0.03, 0.06 and 0.12: rare combinations of cascades and nuisance alarms fill the space between the cores. 34
4.9 NBVFS-WN of the industrial alarm log: the extension tree of the most frequent start event, O1084. 35
4.10 NBVFS-CM (left) and NBVFS-TM (right) of the 421 frequent sequences of the alarm log, the three most frequent start events highlighted. 35
4.11 The fused map: similarity network fusion of the confidence, transaction and content similarities of the synthetic alarm floods. 36
4.12 Louvain communities of the transaction similarity graph: candidate sub-processes. 36
4.13 The sub-log of the first community, the floods in which the pump and the compressor cascades occur together, drawn as a directly-follows graph (chapter 6). 37

5.1 A prefix tree, its continuation entropies and its icicle. 40
5.2 The first five plies of 2,000 games from an opening book: rings are plies, angles are numbers of games. 43
5.3 The opening tree as an icicle: columns are plies, heights are numbers of games, empty space marks games that leave the tree. 44
5.4 Sunbursts of the opening tree at depths 2, 4 and 6. 44
5.5 The continuation-entropy tree of a thousand web-shop journeys: green nodes have a predictable next page, vermilion nodes are decision points. 45
5.6 The double tree of the cart page: the steps before (left, read backwards) and after (right) the first visit to the cart. 45
5.7 The double tree of the motif E–D–C in the phrases of thirteen songs. 46
5.8 A longer pivot: the context of ⟨product cart⟩. 46
5.9 The contrast sunburst of desktop (blue) and mobile (vermilion) visitors. 47
5.10 The contrast sunburst of converted (blue) and lost (vermilion) visitors, first three steps. 47
5.11 LifeFlow of the daily routines: every activity at the mean time since waking up (hours). 48
5.12 LifeFlow per persona: commuters, remote workers and early birds. 49

6.1 Sankey diagram of a thousand web-shop journeys, six pages per step; white boxes collect the visitors who leave. 55
6.2 The four steps after the first visit to the cart (anchored Sankey diagram). 55
6.3 Three and eight pages per step: the merged other band shrinks, the diagram grows busier. 56
6.4 The directly-follows graph of the web shop with its 24 strongest transitions; the two white nodes mark the start and the end of the journeys. 56
6.5 The row-normalized transition matrix of the web shop. 57
6.6 MatrixWave: one transition matrix per step, for the first three steps. 57
6.7 The ten and the forty strongest transitions. 57

6.8 The chord diagram of the transitions of three hundred days, ribbons colored by their source activity. 58
6.9 One chord diagram per persona: commuters, remote workers and early birds. 58
6.10 The growth Sankey of the frequent sequences that start with wake up: columns are lengths, ribbons carry supports, stop collects the support not continued by a drawn child. 59
6.11 The growth Sankey of two start activities, work and lunch. 59
6.12 The pattern-induced higher-order network of the daily routines: 19 activities and 30 memory nodes (ringed), such as work|commute. 60
6.13 The higher-order network of the web-shop journeys: the first-order generator leaves no memory node. 61

7.1 Seriated pattern–sequence incidence matrix of the synthetic alarm floods: four blocks, four cascades. 67
7.2 The seriated transaction similarity of the 40 patterns with its dendrogram. 67
7.3 The seriated content similarity (longest common subsequence) of the same patterns. 68
7.4 Grouped matrix of the sequential rules of the alarm floods: antecedents in columns, consequents in rows, balloon area = support, color = lift. 68
7.5 The same rules colored by confidence. 69
7.6 The click-stream patterns by relative support and confidence; grey lines join every pattern to its parent. 69
7.7 The click-stream patterns of length at least two by confidence and lift, colored by start event. 70
7.8 The pattern landscape of the synthetic alarm floods: support-weighted density of the transaction map, peaks named after their most supported long patterns. 70
7.9 Landscapes with bandwidths 0.15, 0.3 and 0.6. 71
7.10 The containment lattice of the daily-routine patterns: one row per length, edges to one-item super-patterns, closed patterns ringed. 71
7.11 UpSet plot of the supporting days of the seven most supported patterns of length at least two. 72

8.1 Index plot of eighty alarm floods sorted by two highlighted patterns, whose earliest occurrences are marked. 77
8.2 The same floods sorted by length. 78
8.3 The state distribution plot (chronogram) of the daily routines, with the cross-sectional entropy (line, right axis). 78
8.4 The ten most frequent complete web-shop journeys, bar heights proportional to their frequency. 79
8.5 Dotted chart of 120 alarm floods (time since the start of each flood); the line follows the pump cascade. 79
8.6 Time spans of the occurrences of the most frequent patterns. 80

8.7 Sequence logo of the web-shop journeys around the first visit to the cart; heights in bits. . . . 80
8.8 The first notes of the phrases of thirteen songs (probabilities). . . . 80
8.9 Arc diagram of *Ode to Joy*: repeats of at least three notes. . . . 81
8.10 Arc diagram of *Twinkle, Twinkle, Little Star*. . . . 81
8.11 Minimal occurrences of the eight most frequent serial episodes of the busiest day of the alarm log (window ten minutes); the numbers are WINEPI frequencies. . . . 82
8.12 The MDL cover of forty days: every run of activities is colored by the summary pattern that covers it. . . . 82

9.1 How data enter and leave spmviz. . . . 84

10.1 The SIGN benchmark: the CM map of its frequent sequences of length at least two (left) and the sunburst of the first three signs of its 730 utterances (right). . . . 92
10.2 The first five steps of the 343 sessions as a Sankey diagram. . . . 93
10.3 The directly-follows graph of the 14 services. . . . 93
10.4 The continuation-entropy tree of the first four steps. . . . 94
10.5 What comes next after ⟨Your last employer The dismissal⟩: the extensions of the longest matching frequent sequence, with their confidences and supports. . . . 94
10.6 Frequent scenarios starting with the operator action O1084: the three most frequent continuations of every node, edge width proportional to the support. . . . 95
10.7 Online prediction after ⟨O1084 O1085⟩: the extensions of the longest matching frequent sequence ranked by confidence. . . . 96
10.8 A procedure for the analysis of a new log. . . . 96

CHAPTER

1



Introduction

> This chapter explains why frequent sequences need visualization, which questions the different views answer, and how an analysis proceeds from an event log to a picture that can be interpreted and shared. It introduces the network-based visualization of frequent sequences that gave rise to the book, the `spmviz` package, and a first session with it. The theory starts in chapter 2.

1.1 Problem and idea

An event log records what happened, in which order, and to whom: the pages a visitor opened, the alarms a plant raised and the actions its operators took, the activities of a day. Grouping the events by case (a session, a trace, a day) gives a database of sequences. Sequential pattern mining, introduced by Agrawal and Srikant [6], finds the sequences of events that occur, in this order but not necessarily consecutively, in at least a given share of the cases. Thirty years of research have made the mining fast [7, 8, 9, 10, 11] and have produced condensed representations, constraints and interestingness measures [12, 13].

The output of a mining run is a list of patterns with their supports, and the list is long. The four short sequences of the PrefixSpan paper already contain 53 frequent sequences at a support threshold of one half (section 2.6), and a real log yields hundreds at the thresholds that analysts use. The list is also redundant: every subsequence of a frequent sequence is frequent, so a single long pattern appears together with all of its prefixes and many of its subsequences. A table of patterns answers the question *is this sequence frequent?*, but not the questions an analyst actually has. Where does the process branch? Which sequences grow out of which? Which patterns describe the same sub-process? How do two groups of cases differ? What is likely to happen next?

Visualization addresses these questions by exploiting the structure that the patterns share. Frequent sequences form a tree under extension, they inherit their supports from each other, they are supported by overlapping sets of cases, and they can be laid back onto the sequences they were mined from. Each of these relations suggests a family of views, and the literature of visual analytics [14, 15, 16], process mining [17] and social-science sequence analysis [18, 19] has developed them largely independently of each other. The network-based visualization of frequent sequences (NBVFS) of Bántay and Abonyi [1] adds three views that make the relations between the frequent sequences themselves visible: a weighted network of extensions and two maps that place similar sequences close to each other. Figure 1.1 shows how the pieces fit together.

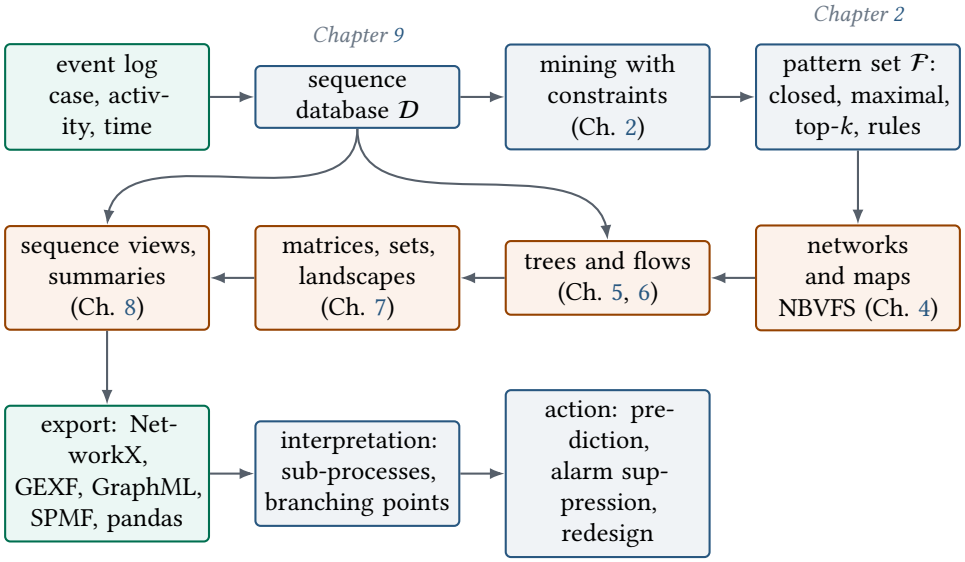


Figure 1.1. From an event log to an interpretable picture. The log becomes a sequence database; mining with constraints gives a pattern set, which can be condensed further. Some views summarize the raw sequences directly (sunbursts, Sankey diagrams, index plots), others draw the pattern set (networks, maps, lattices), and several do both (sequence index plots with pattern occurrences). Every view exports its data, so the analysis can continue in other tools.

1.2 What visualization can and cannot tell

The views of this book are descriptive. They show which sequences are frequent and how the frequent sequences relate to each other, but frequency is not causality, and a frequent sequence of two alarms does not prove that the first causes the second. Three further limits should be kept in mind.

First, every view depends on the mining parameters. The support threshold, the gap and span constraints and the maximum length decide which patterns exist

at all, and a pattern absent from a picture may only be absent from the pattern set. Chapter 2 shows how quickly the number of patterns changes with the threshold and the gap, and each example of the book ends with such a parameter study. Second, many views aggregate. A Sankey diagram or a directly-follows graph keeps only consecutive pairs and forgets the history; a sunburst keeps the history but only from the start of the sequence. The higher-order networks of chapter 6 and the word trees of chapter 5 relax these limitations, and the network-based maps of chapter 4 keep the patterns themselves as the objects of the picture. Third, maps obtained by multidimensional scaling distort distances. Their stress, reported by the package with every map, tells how much [20].

1.3 The toolbox

`spmviz` is a pure Python package built on NumPy, SciPy, pandas, NetworkX and Matplotlib [21, 22, 23, 24]. Its modules follow the workflow of fig. 1.1:

- *data*: `SequenceDatabase` reads event tables, SPMF files, XES logs and pm4py objects, and `PatternSet` holds patterns with their supports and supporting sequences (`spmviz.data`, `spmviz.patterns`, `spmviz.io`);
- *mining*: PrefixSpan, SPAM, CM-SPAM, SPADE and GSP with gap, span and time constraints, closed, maximal and generator patterns, top-k mining, sequential rules, episodes, and adapters to the `prefixspan`, `seq2pat` and `gsppy` packages and to SPMF [25, 26, 27, 28] (`spmviz.mining`, `spmviz.backends`);
- *structure*: measures, similarities, extension networks, prefix trees, transition graphs, higher-order networks, communities, embeddings, MDL summaries and next-event prediction (`spmviz.measures`, `spmviz.networks`, `spmviz.embedding`, `spmviz.summary`, `spmviz.predict`);
- *views*: the `NBVFS` estimator, nearly forty static views in `spmviz.viz` and interactive twins in `spmviz.interactive`.

The package bundles the two real logs of the NBVFS paper, the anonymized alarm log of an alkylation plant and a click-stream excerpt of the BPI Challenge 2016 [29], small textbook databases, a public-domain melody collection and a chess-opening book, generators of synthetic data with known structure, and a downloader for the benchmark databases distributed with SPMF.

1.4 A first session

The click-stream of the NBVFS paper serves as the running example of the book: 343 sessions of applicants for unemployment benefit on the portal of the Dutch Employee Insurance Agency, over 14 services coded 0–13. Mining it as in the paper and drawing the extension network of the sessions that start with service 2 (Hours worked) takes four lines:

```
import spmviz as sv

db = sv.datasets.load_clickstream()                    # 343 sessions,
    14 services
ps = sv.mine(db, min_support=0.01, max_gap=1, max_length=10)
print(ps)                    # PatternSet(n_patterns=241,
    algorithm='cmspam', ...)
wn = sv.NBVFS("wn", start_items=[2]).fit(ps)            # weighted
    extension network
wn.plot()                                               # wn.graph_ is
    a networkx.DiGraph
sv.predict_next(ps, [0, 1])   # after "Your last employer -> The
    dismissal": The dismissal (0.45), ...
```

The pattern set keeps every pattern with its support and the identifiers of its supporting sequences; it can be filtered, condensed (`ps.closed()` leaves 219 of the 241 patterns, `ps.maximal()` 53), turned into a table (`ps.to_frame()`) or a graph (`ps.to_networkx()`), and drawn by every view of the book.

1.5 How the book is organized

Chapter 2 defines sequences, containment, support and constraints, the condensed representations of a pattern set, the extension tree and its confidences, and the five mining algorithms of the package. Chapter 3 surveys the visualizations of frequent sequences proposed in the literature and arranges them by the relation they draw and the question they answer. Chapter 4 develops the network-based views of Bántay and Abonyi and extends them with fused similarities and community-based segmentation. Chapter 5 presents trees of prefixes and patterns, among them two new views, the continuation-entropy tree and the contrast sunburst. Chapter 6 covers flows and transitions, from Sankey diagrams to pattern-induced higher-order networks. Chapter 7 treats matrices, maps and set views, including pattern landscapes. Chapter 8 returns to the sequences themselves: index plots, chronograms, dotted charts, sequence logos, arc diagrams, episodes and summaries by the minimum description length principle. Chapter 9 shows how data enter and results leave the package, and chapter 10 reports benchmarks and two case studies. Chapter A lists the application programming interface, chapter B maps the scripts of the original paper to the package, and chapter C catalogues the example scripts.

1.6 Further reading

The surveys of Mooney and Roddick [12] and of Fournier-Viger and co-authors [13] cover the algorithms of sequential pattern mining. Guo and co-authors [14] and Yeshchenko and Mendling [15] survey the visual analysis of event sequences, and Jentner and Keim [16] the visualization of patterns. Van der Aalst's textbook [17] is the standard reference of process mining, and Gabadinho and co-authors [18] describe the sequence-analysis tradition of the social sciences and its R package TraMineR.

CHAPTER

2



Sequences, patterns and their mining

> Every view of the book draws either a database of sequences or a set of frequent sequences mined from it, so this chapter fixes the vocabulary: sequences of itemsets, containment under gap, span and time constraints, support, the condensed representations of a pattern set, and the extension tree whose confidences the network-based views draw. It then explains the four mining strategies implemented in `spmviz` — pattern growth, vertical bitmaps, vertical id-lists and level-wise search — and closes with the data model of the package and examples on small and everyday data.

2.1 Problem and idea

A customer buys several items at once, then returns a week later; an operator acknowledges two alarms in the same second. Such data are sequences of *itemsets*: sets of events that happen together, ordered in time. A pattern is itself a sequence of itemsets, and the basic question of sequential pattern mining is in how many sequences of the database a pattern occurs, its elements in the right order, each element contained in a distinct element of the sequence [6]. The question has two refinements that matter in practice. Occurrences can be restricted, for instance to consecutive elements or to a time window, because a pattern that joins events hours apart may describe nothing real [7]. And the answer is wanted for all patterns at once that occur in enough sequences, which is a search problem over an exponentially large space, made tractable by the fact that a pattern cannot be more frequent than its prefixes.

2.2 Theory

2.2.1 Sequences and containment

Let $\mathcal{I}$ be a finite set of items (events, pages, alarms). An *itemset* is a non-empty subset of $\mathcal{I}$, a *sequence* $s = \langle s_1 s_2 \cdots s_n \rangle$ an ordered list of itemsets, its *elements*; the *length* of s is the total number of items $\sum_j |s_j|$. When every element is a single item we write $\langle a\ b\ c \rangle$; parentheses group simultaneous items, as in $\langle a\,(b\,c)\,d \rangle$. A *sequence database* $\mathcal{D} = (s^{(1)}, \ldots, s^{(N)})$ is a list of sequences, each with an identifier and possibly a timestamp t_j for each element and attributes (device, group, outcome) for the whole sequence.

A sequence $\varphi = \langle \varphi_1 \cdots \varphi_m \rangle$ is *contained* in s, written $\varphi \sqsubseteq s$, if there are indices $j_1 < j_2 < \cdots < j_m$ with $\varphi_k \subseteq s_{j_k}$ for every k; the indices form an *occurrence* of φ. Constraints restrict the admissible occurrences [7, 26]:

- a *gap* constraint bounds the distance of consecutive matched elements, $g_{\min} \le j_{k+1} - j_k \le g_{\max}$; following the convention of SPMF [28], $g_{\max} = 1$ requires adjacent elements and $g_{\max} = 2$ allows one element in between;
- a *span* constraint bounds the whole occurrence, $j_m - j_1 \le w$;
- with timestamps, both bounds can be applied to $t_{j_{k+1}} - t_{j_k}$ and $t_{j_m} - t_{j_1}$ instead of the indices.

The *support* $\sigma(\varphi)$ is the number of sequences of $\mathcal{D}$ that contain φ under the constraints, and the *relative support* is $\sigma(\varphi)/N$. Given a threshold $\sigma_{\min}$, the set of *frequent sequences* or *sequential patterns* is $\mathcal{F} = \{\varphi : \sigma(\varphi) \ge \sigma_{\min}\}$. Figure 2.1 shows the four-sequence database of the PrefixSpan paper [30], the running toy example of this chapter.

2.2.2 Anti-monotonicity and the extension tree

Without constraints, every subsequence of a sequence contained in s is contained in s as well, so $\varphi' \sqsubseteq \varphi$ implies $\sigma(\varphi') \ge \sigma(\varphi)$: support is *anti-monotone*, and $\mathcal{F}$ is closed under taking subsequences. This is the property that makes mining feasible. Gap constraints weaken it: deleting a middle element of an occurrence with $g_{\max} = 1$ leaves a gap of two, so a subsequence of a frequent sequence need not be frequent. Prefixes survive, however, because truncating an occurrence after its k-th element keeps every gap and shortens the span. Under all constraints of section 2.2.1 the set $\mathcal{F}$ is therefore *prefix-closed*.

The *parent* $\pi(\varphi)$ of a pattern is the pattern without its last item; conversely φ is a one-item *extension* of $\pi(\varphi)$, either a *sequence extension* that appends a new element, $\langle a\ c \rangle \to \langle a\ c\ b \rangle$, or an *itemset extension* that adds an item to the last element, $\langle a\ b \rangle \to \langle a\,(b\,c) \rangle$ [8, 10]. The parent relation turns a prefix-closed $\mathcal{F}$ into a forest whose roots are the frequent single items: the *extension tree*. Its edges

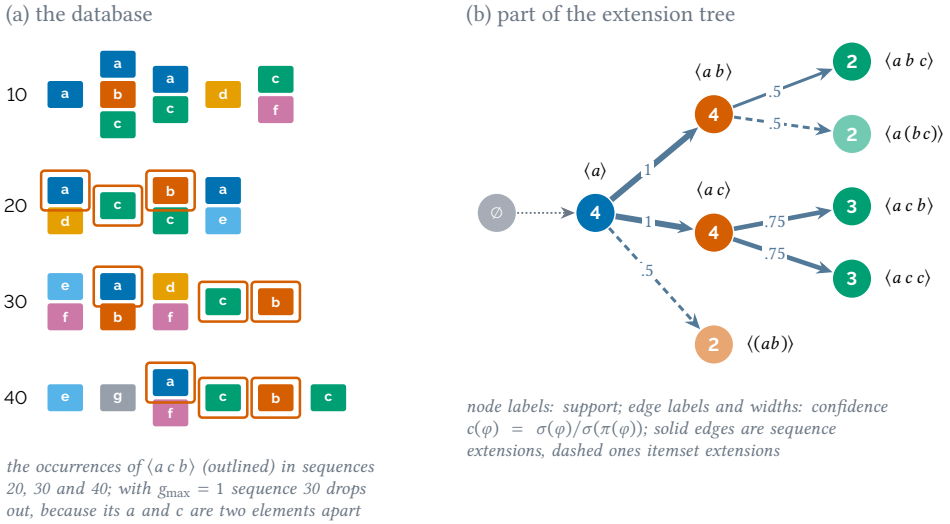


Figure 2.1. The database of the PrefixSpan paper [30] and part of its extension tree at $\sigma_{\min} = 2$. (a) Simultaneous items are stacked; each color is one item; the outlined tiles are occurrences of $\langle a\ c\ b \rangle$. (b) Every frequent sequence is the child of its prefix; the edge carries the confidence of the extension.

carry the *confidence* of the extension, the share of the sequences containing the parent that also contain the child,

$$c(\varphi) = \frac{\sigma(\varphi)}{\sigma(\pi(\varphi))} \in (0, 1], \qquad c^*(\varphi) = \frac{\sigma(\varphi)}{\sigma(\langle y_1 \rangle)} = \prod_{\psi} c(\psi), \tag{2.1}$$

where y_1 is the first item of φ and the product runs over the patterns on the path from $\langle y_1 \rangle$ to φ. The *cumulative confidence* $c^*(\varphi)$ is the share of the sequences starting the tree that reach φ. Bántay and Abonyi draw exactly this tree, with these weights, in their weighted network view (chapter 4) [1]. The confidence of the last extension is related to classical rule measures: if y is the added item, the *lift* $c(\varphi)\, N/\sigma(\langle y \rangle)$ compares it with the frequency of y alone, and lifts well above one mark extensions that are more than coincidences.

2.2.3 Condensed representations

A frequent sequence of length ℓ without constraints implies that all its $2^\ell - 1$ non-empty subsequences are frequent, so $\mathcal{F}$ explodes when long patterns exist. Three condensed sets are standard [31, 32, 33, 34, 35]:

- a pattern is *closed* if no super-pattern has the same support; the closed patterns determine the support of every frequent pattern, so the representation is lossless;
- a pattern is *maximal* if no super-pattern is frequent; the maximal patterns determine which patterns are frequent, but not their supports;

Algorithm 2.1 Depth-first pattern growth

Input: database $\mathcal{D}$, threshold $\sigma_{\min}$, constraints, maximum length L
Output: all frequent sequences $\mathcal{F}$ with supports and supporting sequences

```
1 F ← ∅;
2 foreach item y with σ(⟨y⟩) ≥ σ_min do
3     Grow(⟨y⟩)
4 procedure Grow(φ)
5     add φ to F;
6     if length of φ = L then return;
7     compute σ of every sequence extension φ ⋄_s y and itemset
        extension φ ⋄_i y;
8     foreach extension ψ with σ(ψ) ≥ σ_min do
9         Grow(ψ)
```

PrefixSpan scans the projected database of φ, the suffixes after the earliest occurrence of each sequence; with gap constraints it keeps all occurrence end points. *SPAM* and *CM-SPAM* keep a bitmap of the positions where φ can end and obtain an extension by a shift and an AND; CM-SPAM prunes candidates whose two-item pattern is infrequent (co-occurrence map). *SPADE* joins the id-lists of two patterns that share a prefix. *GSP* generates the candidates of length $\ell + 1$ from the frequent patterns of length ℓ and counts them in one scan.

- a pattern is a *generator* if no sub-pattern has the same support; generators are the minimal patterns of their support class.

When no sensible threshold is known, *top-k* mining returns the k most frequent patterns, raising the threshold internally as better patterns are found [36]. Other generalizations of the pattern concept appear later in the book: sequential rules, which split a pattern into an antecedent and a consequent (chapter 7) [37], and episodes, frequent patterns of one long event stream without case boundaries (chapter 8) [38].

2.3 Mining algorithms

All exact miners of the package enumerate the extension tree depth first or level by level and stop at infrequent nodes; prefix-closure guarantees that nothing frequent lies below them. They differ in how they compute the supports of the candidate extensions (algorithm 2.1).

PrefixSpan [8, 30] grows patterns in projected databases: the sequences that contain φ, cut after the place where an occurrence of φ ends, contain every frequent extension of φ in their suffixes. The package uses pseudo-projection (pointers

instead of copies) and, when a gap or span constraint is set, a map from each possible end point of an occurrence to its latest start, which is all a constrained extension needs. SPAM [10] represents a pattern by one bit per element of every sequence, set where an occurrence of the pattern can end. A sequence extension by y is the bitmap of φ shifted to the right (all positions after the first set bit, or only the next $g_{\max}$ positions under a gap constraint) and intersected with the bitmap of y; an itemset extension is a plain intersection. In Python the bitmaps of all sequences are packed into one arbitrary-precision integer, so the shift and the intersection run as single machine-level operations. CM-SPAM [11], the algorithm used in the NBVFS paper, adds a co-occurrence map that discards extensions whose two-item pattern $\langle y' \; y \rangle$ (or $\langle (y' \; y) \rangle$) is infrequent. SPADE [9] stores for every pattern the list of (sequence, end position) pairs and joins two lists to obtain the occurrences of an extension. GSP [7] is the level-wise, Apriori-style algorithm: it is the slowest of the five but the simplest to reason about, and it serves as an independent check.

The five algorithms return identical pattern sets. The test suite of the package compares them with each other and with a brute-force enumeration on random databases with itemsets, gaps and timestamps, and section 2.6 compares them on synthetic click streams. Their speeds differ. On the benchmark databases of chapter 10 the bitmap algorithms, SPAM and CM-SPAM, are the fastest without constraints, because a Python integer operation processes all sequences at once; with a gap constraint PrefixSpan and CM-SPAM are comparable, and GSP is 30 to 370 times slower.

2.4 The data model of spmviz

Two classes carry the data of every analysis. A `SequenceDatabase` stores sequences of itemsets as tuples of tuples, with identifiers, optional timestamps per element, optional attributes per sequence (a pandas data frame) and optional display labels for the items. It is built from Python lists, strings, wide or long tables, event logs with case, activity and timestamp columns, SPMF files, XES files and pm4py logs (chapter 9). A `PatternSet` stores the patterns in the same canonical form together with their supports, the identifiers of their supporting sequences, the database and the mining parameters. Because the supporting sequences are kept, every measure and view can be computed afterwards without mining again: confidences, closed and maximal subsets, incidence and similarity matrices, rules, contrasts between groups.

Patterns can be written in several ways. `"a (b c) d"` separates items by spaces and groups simultaneous items in parentheses; the arrow notation `"wake up -> coffee -> (tv, read)"`, the default output of `format_pattern`, allows items with spaces; `"1 -1 2 3 -1"` is the SPMF format. `db.pattern(...)` resolves a pattern written with the labels of a database to its items. Table 2.1 summarizes the notation of the book and its counterparts in the code.

Table 2.1. Notation of the book, the docstrings and the code.

Symbol	Meaning	Code
$\mathcal{I}$	item alphabet	`db.alphabet`
$\mathcal{D}$, N	sequence database and its number of sequences	`SequenceDatabase`, `len(db)`
$s = \langle s_1 \cdots s_n \rangle$	sequence of itemsets	`db[i]` (tuple of tuples)
$\varphi \sqsubseteq s$	φ is contained in s (under the constraints)	`db.contains(p)`
$g_{\min}$, $g_{\max}$, w	gap and span bounds (index or time units)	`min_gap`, `max_gap`, `max_span`, `gap_unit`
$\sigma(\varphi)$, $\sigma(\varphi)/N$	support and relative support	`ps.support`, `rel_support`
$\mathcal{F}$	set of frequent sequences	`PatternSet`
$\pi(\varphi)$	parent: φ without its last item	`parent_pattern(p)`
$c(\varphi)$, $c^*(\varphi)$	confidence and cumulative confidence, eq. (2.1)	`ps.confidence()`, `cum_confidence()`
$\mathbf{T}$	pattern–sequence incidence matrix	`ps.incidence()`
$\mathbf{A}$, $\mathbf{A}^*$	confidence matrix and its max-product closure (chapter 4)	`confidence_matrix`, `transitive_confidence`
$\mathbf{S}$, $\mathbf{D} = \mathbf{1} - \mathbf{S}$	similarity and dissimilarity of patterns	`similarity_matrix`

In a nutshell Sequences and patterns

- A pattern is contained in a sequence if its itemsets are subsets of distinct elements in the same order; gap, span and time constraints restrict the occurrences.
- Support is anti-monotone without constraints; with gap constraints the frequent sequences are still prefix-closed, so they form the *extension tree*.
- Confidence $c(\varphi) = \sigma(\varphi)/\sigma(\pi(\varphi))$ weights the edges of the tree; the cumulative confidence is the product along the path.
- Closed patterns are a lossless summary, maximal patterns a compact one; top-k mining avoids choosing a threshold.
- PrefixSpan, SPAM, CM-SPAM, SPADE and GSP return the same patterns and differ only in speed.

2.5 Using it in spmviz

`sv.mine(db, min_support, ...)` is the single entry point: `min_support` is a share of the sequences (rounded up, as in SPMF) or an absolute count, `algorithm` is `"prefixspan"`, `"cmspam"`, `"spam"`, `"spade"`, `"gsp"` or an external back end, and `"auto"` picks CM-SPAM unless a span or time constraint or top-k mining calls for PrefixSpan. `max_gap`, `min_gap`, `max_span`, `gap_unit`, `min_length` and `max_length` set the constraints, and `closed=True` or `maximal=True` condense the result. `sv.topk(db, k)` returns the k most frequent patterns. The results

compare like Python objects: two pattern sets are identical when their `patterns` and `support` attributes are equal.

```
db = sv.datasets.load_prefixspan_example()
for sid, s in zip(db.ids, db):
    print(sid, sv.format_pattern(s, "angle"))

ps = sv.mine(db, min_support=0.5)          # in at least 2 of the 4
    sequences
print(len(ps), "frequent sequential patterns")
print(ps.to_frame().head(12)[["pattern", "support", "confidence"]])
```

```
print(len(ps.closed()), "closed,", len(ps.maximal()), "maximal")
print(ps.maximal().labels("angle"))
```

2.6 Examples

A textbook database. At $\sigma_{\min} = 2$ the four sequences of fig. 2.1 contain 53 frequent sequences, 17 of them closed and 10 maximal. The glyph table of fig. 2.2 draws every pattern as a row of colored tiles, simultaneous items stacked, with bars for its support and confidence; it is the most compact faithful picture of a short pattern list. Figure 2.3 shows how the set grows when the threshold falls: at a quarter of the sequences there are 812 frequent sequences, but only 21 closed and 4 maximal ones.



Figure 2.2. A glyph table of the longest frequent sequences of the PrefixSpan database: one row per pattern, items as tiles, simultaneous items stacked; the bars give the support and the confidence of the last extension.

An everyday example. The generator `make_daily_routines` produces days of three kinds of people — early birds, commuters and remote workers — as sequences of activities from waking up to going to sleep. Three hundred days have

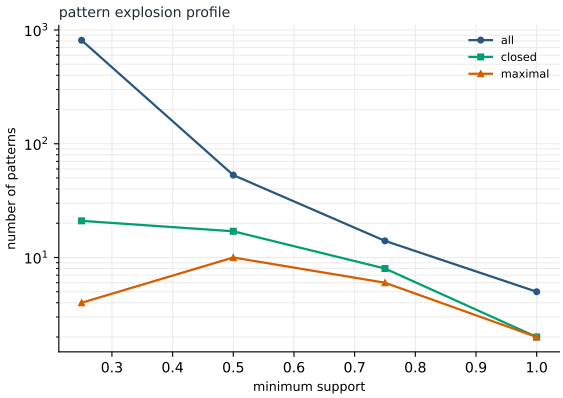


Figure 2.3. The pattern explosion: numbers of all, closed and maximal frequent sequences of the PrefixSpan database against the support threshold.

19 activities and between 9 and 16 steps. The sunburst of fig. 2.4 summarizes how the days begin before anything is mined; fig. 2.5 draws the 96 frequent sequences found at 30% support when one activity may be skipped ($g_{\max} = 2$), as a tree that keeps the three most frequent continuations of every node.

```
db = sv.datasets.make_daily_routines(300, random_state=0)
print(db.stats())
print(sv.format_pattern(db[0]))
plot_sunburst(db, max_depth=4, title="how the days begin")
```

```
ps = sv.mine(db, min_support=0.3, max_gap=2, max_length=6)
print(ps)
plot_prefix_tree(ps, max_depth=4, max_children=3, title="frequent
    sequences as a tree (three most frequent continuations per node)")
```

The gap matters more than the threshold here: with $g_{\max} = 1$ only 25 sequences are frequent, with $g_{\max} = 2$ 96, with $g_{\max} = 3$ 346, and without a gap constraint 1425, most of them joining activities hours apart. The glyph table of the ten most supported patterns of length at least three is fig. 2.6.

Five algorithms, one answer. On a thousand synthetic web-shop journeys (`make_customer_journeys`), the five built-in miners and the `prefixspan` package return the same 890 patterns at 3% support; the run times are compared in fig. 2.7.

```
results, times = {}, {}
for alg in ["prefixspan", "cmspam", "spam", "spade", "gsp"]:
    t0 = time.perf_counter()
    results[alg] = sv.mine(db, min_support=0.03, algorithm=alg,
    max_length=6)
    times[alg] = time.perf_counter() - t0
ref = results["prefixspan"]
for alg, ps in results.items():
```

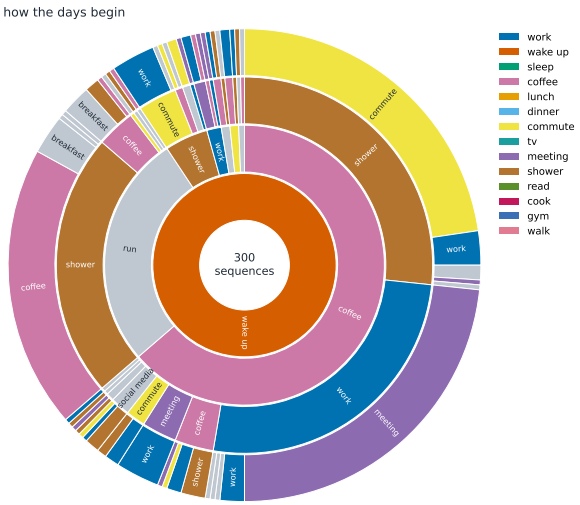


Figure 2.4. How three hundred days begin: the sunburst of the first four activities (rings are steps, angles numbers of days).

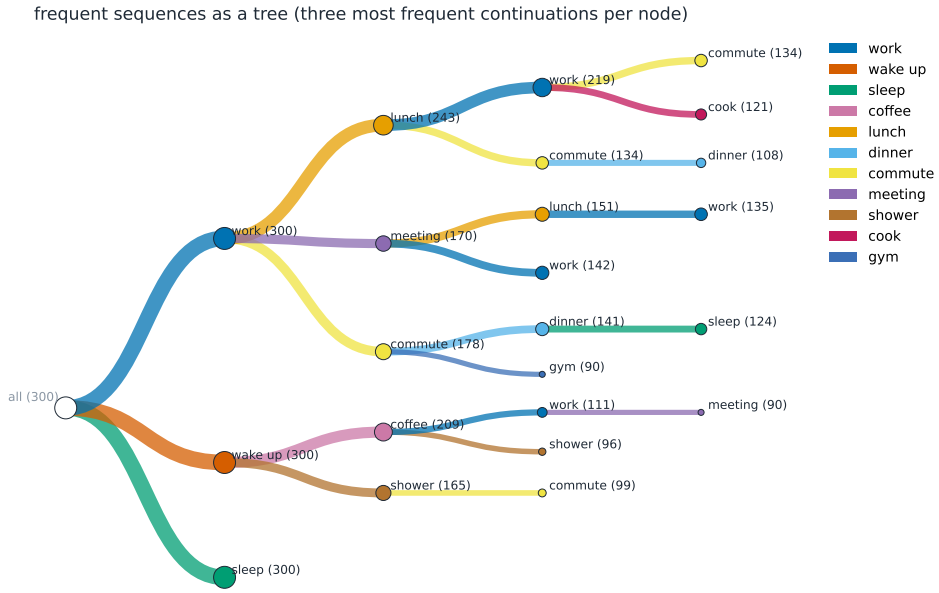


Figure 2.5. The frequent sequences of the daily routines as a pattern tree: edges as wide as the support, three most frequent continuations per node.

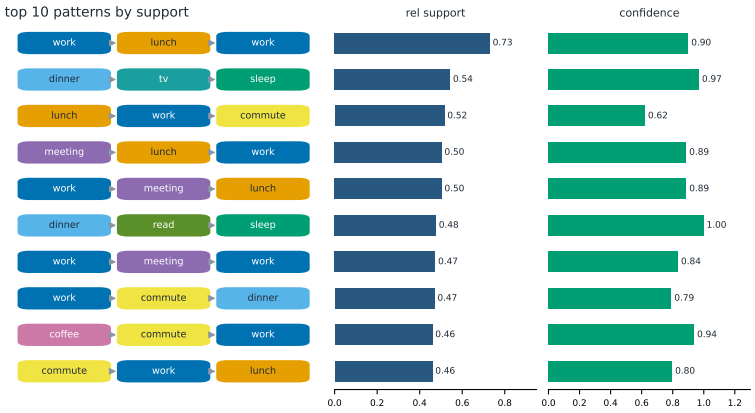


Figure 2.6. The ten most supported daily-routine patterns of length at least three.

```
    same = ps.patterns == ref.patterns and np.array_equal(ps.support,
    ref.support)
    print(f"{alg:10s} {len(ps):5d} patterns  {times[alg]:6.3f} s
    identical={same}")
```

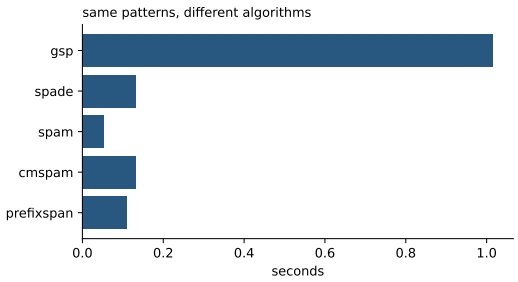


Figure 2.7. Run times of the five built-in miners on a thousand web-shop journeys; all return the same 890 patterns.

Constraints. A pattern such as ⟨wake up coffee⟩ occurs in 276 of the 300 days, in 191 if the two activities must be adjacent, in 151 if coffee must follow within half an hour, and in 268 if the whole occurrence must fit in two hours:

```
p = "wake up -> coffee"            # arrows separate the elements; items
    may contain spaces
for kw in [{}, {"max_gap": 1}, {"max_gap": 0.5, "gap_unit": "time"},
    {"max_span": 2.0, "gap_unit": "time"}]:
    print(f"{str(kw):45s} support = {db.support(p, **kw)}")
```

The sequence index plot of fig. 2.8 (chapter 8) shows where the occurrences are.

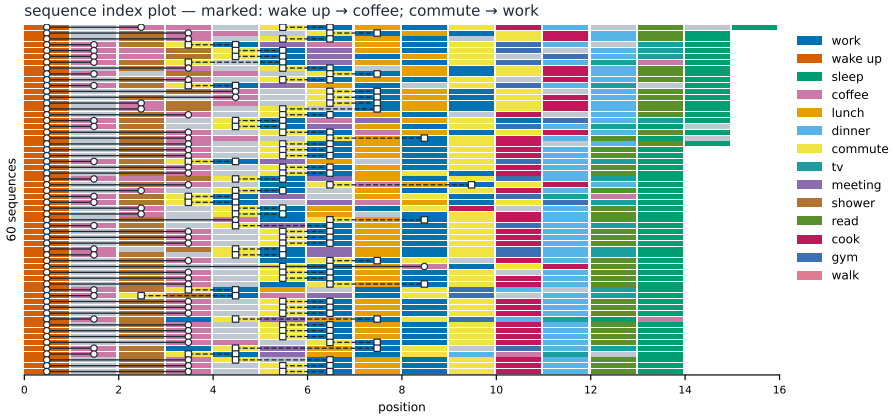


Figure 2.8. Sixty days as rows of colored activities, sorted by the highlighted patterns ⟨wake up coffee⟩ and ⟨commute work⟩, whose earliest occurrences are marked.

CHAPTER

3



A map of the visualizations of frequent sequences

Visualizations of event sequences and of frequent sequences have been proposed in at least four communities: data mining, visual analytics, process mining, and social-science sequence analysis. This chapter surveys them and arranges them by the object they draw — individual sequences, transitions, prefix structures, pattern sets, or the relation between patterns and sequences — and by the question they answer. The arrangement is the plan of the following chapters, and its last table lists, for every technique, its origin and its implementation in `spmviz`.

3.1 Problem and idea

The visual analysis of event sequences has been surveyed several times. Guo and co-authors organize more than a hundred systems by the analysis task (summarization, prediction, anomaly detection, comparison) and by the visual representation [14]; Yeshchenko and Mendling survey event-sequence analysis and visualization across process mining and visual analytics [15]; Jentner and Keim review the visualization of patterns, sequential ones included [16]. The surveys agree on a basic distinction. Some views draw the *sequences* — one row per case, or an aggregation of all cases into a tree or a flow — and let the eye find the patterns. Others draw the *patterns* that an algorithm has found, as a list, a network, a lattice or a map. A third group connects the two by laying patterns back onto the sequences that support them. For an analyst who has mined frequent sequences, a more useful organization starts from the relation the view makes visible, because the relation determines the questions that the view can answer.

3.2 Five objects, five families

Figure 3.1 arranges the techniques by the object they draw, from the individual sequence to the pattern set.

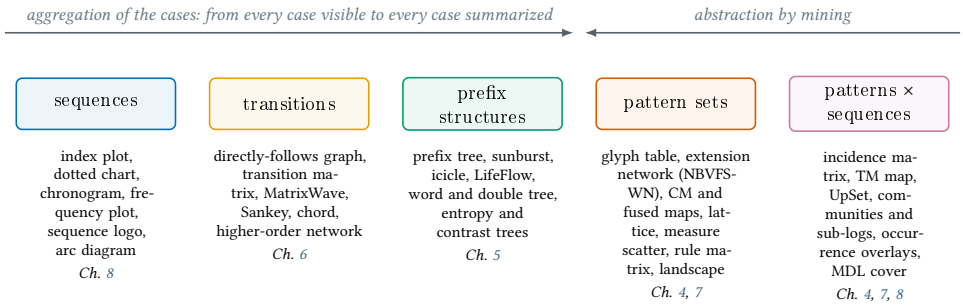


Figure 3.1. Five families of visualizations, arranged by the object they draw. From the left, the views aggregate the cases more and more; from the right, they abstract the data by mining. Pattern-centred views (red and purple) are the subject of most of the new methods of the book.

Sequences. The *sequence index plot* of social-science sequence analysis draws every case as a row of colored cells [18, 19]. It hides nothing, which is its strength for a few hundred cases and its weakness for more; sorting the rows (by a pattern, by length, by a clustering) decides what can be seen. The *state distribution plot* or chronogram stacks the states seen at each position, the *sequence frequency plot* the most frequent complete sequences [18], and the relative frequency sequence plot of Fasang and Liao draws representative sequences of equal-sized groups [39]. Process mining contributed the *dotted chart*, which places every event at its time and case [40], bioinformatics the *sequence logo*, which stacks the symbols of aligned positions with heights proportional to their information content [41], and text visualization the *arc diagram*, which joins repeated subsequences of one long string [42].

Transitions. The simplest aggregation keeps consecutive pairs only. The *directly-follows graph* of process mining [17] and the first-order Markov chain are the same object; drawn as a matrix it becomes the transition matrix, drawn step by step it becomes MatrixWave, a chain of transition matrices between consecutive positions [43], and drawn on a circle it becomes a chord diagram [44]. Sankey or alluvial diagrams follow the cases from step to step [45]; Frequence used them to show frequent event sequences of patients [46], and DecisionFlow extended them to thousands of event types [47]. Sequence braiding draws the flows of sequences together with their attributes [48]. All first-order views forget where a case came from. Higher-order networks keep this memory where it matters: Rosvall and co-authors showed that second-order Markov dynamics change the communities found in flow networks [49], and Xu, Wickramarathne and Chawla introduced variable-order networks in which a node may carry its recent history [50].

Prefix structures. Merging the common prefixes of the sequences gives a tree, the oldest aggregation of event sequences. The *sunburst* draws it as rings [51], the *icicle* as stacked bars [52]; Session Viewer applied such views to web sessions [53], LifeFlow placed every node at the mean time since the start [54], EventFlow simplified the tree by merging rare branches [55], and CoreFlow extracted the branching patterns from it [56]. The *word tree* of Wattenberg and Viégas grows the tree from a chosen word [57]; the *double tree* grows it in both directions [58], and Krause and co-authors enriched double trees of event sequences with category-based comparison [59]. ActiviTree explores activity sequences by graph similarity [60].

Pattern sets. Systems that mine first and draw afterwards usually show the patterns as a list of glyphs, linked to other views: Frequence [46], the clinical event-pattern methodology of Gotz, Wang and Perer [61], the clickstream explorer of Liu and co-authors [62], and the pattern-growth explorer of Vrotsou and Nordman, which lets the analyst steer the mining itself [63]. Stolper, Perer and Gotz plotted the patterns as points in the plane of two measures while the mining was still running [64]. Association-rule visualization contributed the grouped matrix of Hahsler and Karpienko [65, 66], and set visualization the UpSet plot [67]. Summaries by the minimum description length principle select a few patterns that compress the data [68, 69, 70], and Sequence Synopsis draws such a summary as a set of representative paths [71]. The network-based views of Bántay and Abonyi draw the pattern set itself as a weighted network of extensions and as two maps [1].

Patterns and sequences. The relation between a pattern set and the cases is a binary matrix, the pattern–sequence incidence matrix. Seriated, it shows blocks of patterns supported by the same cases [72]; summarized as a similarity it becomes the transaction map of NBVFS; and laid onto an index plot or a dotted chart it shows where a pattern occurs. For dense relations, matrix views are more readable than node-link diagrams [73].

3.3 Which view answers which question

The questions of an analysis are more stable than the views. Table 3.1 arranges the views of the book by the questions they answer best; most questions have more than one good answer, and the examples of the following chapters show the same data in several views.

Table 3.1. Questions about event sequences and the views that answer them.

Question	Views (chapter)
How do the cases begin, and how do they branch?	sunburst, icicle, prefix tree, Sankey (5, 6)
Where is the next step unpredictable?	continuation-entropy tree, prediction view (5, 10)
What happens before and after an event?	word tree, double tree, sequence logo, anchored Sankey (5, 8, 6)
How long do the steps take?	LifeFlow icicle, dotted chart, span box plots (5, 8)
Which event follows which?	directly-follows graph, transition matrix, MatrixWave, chord (6)
Does the next step depend on the history?	higher-order network (6)
Which frequent sequences grow out of which?	extension network NBVFS-WN, growth Sankey, lattice (4, 6, 7)
Which patterns belong to the same scenario or sub-process?	NBVFS-CM and -TM maps, fused map, communities, incidence and similarity matrices, landscape (4, 7)
Which patterns are interesting rather than merely frequent?	measure scatter, grouped rule matrix (7)
How do two groups of cases differ?	contrast sunburst, contrast table, per-group chords (5, 6)
Where in the cases does a pattern occur?	index plot and dotted chart with highlighted patterns, MDL cover (8)
What is the shortest faithful summary?	closed and maximal sets, MDL summary, glyph table (2, 8)

3.4 The catalogue

Table 3.2 lists the techniques implemented in `spmviz` with their origin. Views marked *new* were developed for this book; their construction is explained in the chapter given in the last column.

Table 3.2. Visualizations of frequent sequences and their implementation in `spmviz`.

Technique	Origin	`spmviz` function	Ch.
glyph table of patterns	[46, 62]	`plot_patterns`	2
pattern explosion curve	[31]	`plot_explosion`	2
weighted extension network (WN)	[1]	`NBVFS("wn")`, `plot_nbvfs_wn`	4
confidence-based MDS map (CM)	[1]	`NBVFS("cm")`, `plot_nbvfs_cm`	4
transaction-based MDS map (TM)	[1]	`NBVFS("tm")`, `plot_nbvfs_tm`	4
fused similarity map	new, after [74]	`NBVFS("fused")`	4

Technique	Origin	`spmviz` function	Ch.
pattern communities and sub-logs	new, after [75]	`pattern_communities`, `sublogs`, `plot_communities`	4
prefix (pattern) tree	[53, 56]	`plot_prefix_tree`	5
sunburst	[51]	`plot_sunburst`	5
icicle, LifeFlow	[52, 54]	`plot_icicle`	5
word tree, double tree	[57, 58, 59]	`plot_double_tree`	5
continuation-entropy tree	new	`plot_prefix_tree(color_by="entropy")`	5
contrast sunburst and table	new, after [76]	`plot_sunburst(color_by="contrast")`, `measures.contrast`	5
Sankey (alluvial) diagram	[45, 46]	`plot_sankey`	6
directly-follows graph	[17]	`plot_transition_graph`	6
transition matrix, MatrixWave	[43]	`plot_transition_matrix`, `plot_matrixwave`	6
chord diagram	[44]	`plot_chord`	6
growth Sankey of the extension tree	new	`plot_growth_sankey`	6
pattern-induced higher-order network	new, after [50]	`higher_order_network`, `plot_hon`	6
pattern–sequence incidence matrix	[72]	`plot_incidence`	7
seriated similarity matrix	[72, 73]	`plot_similarity_matrix`, `seriate`	7
grouped matrix of sequential rules	[65]	`plot_rules_matrix`	7
measure scatter plot	[64]	`plot_pattern_scatter`	7
pattern landscape	new	`plot_landscape`	7
lattice (Hasse diagram)	[31]	`plot_lattice`	7
UpSet plot	[67]	`plot_upset`	7
sequence index plot	[18, 19]	`plot_sequence_index`	8
state distribution plot (chronogram)	[18]	`plot_state_distribution`	8
sequence frequency plot	[18]	`plot_sequence_frequency`	8
dotted chart	[40]	`plot_dotted_chart`	8
gap and span distributions	new	`plot_gap_distribution`	8
sequence logo	[41]	`plot_sequence_logo`	8
arc diagram	[42]	`plot_arc_diagram`	8
episode timeline	[38]	`plot_episode_timeline`	8
MDL cover plot	[69, 71]	`plot_cover`	8
next-event prediction view	[3, 37]	`plot_prediction`	10
overview dashboard	new	`plot_overview`	10

3.5 Further reading

Guo and co-authors [14] give the most systematic review of the visual analysis of event sequences, with a taxonomy of tasks; Yeshchenko and Mendling [15] connect it to process mining. Behrisch and co-authors [72] review the seriation of matrices, which underlies several views of chapter 7, and Borg and Groenen [20] the multidimensional scaling behind the maps of chapter 4.

CHAPTER

4



Network-based visualization of frequent sequences

This chapter develops the three network-based views of frequent sequences proposed by Bántay and Abonyi [1]: the weighted extension network (NBVFS-WN), which shows how frequent sequences grow out of each other; the confidence-based map (NBVFS-CM), which places the sequences of the same scenario close together; and the transaction-based map (NBVFS-TM), which groups the sequences that occur in the same cases. It then answers two questions left open by the paper, how to combine the similarities and how to segment the pattern set, with a fused map and with communities that cut the log into sub-logs. The chapter closes with the reproduction of the paper on its two logs.

4.1 Problem and idea

The operators of a chemical plant face alarm floods: dozens of alarms in minutes, some caused by others, interleaved with their own actions. Frequent sequences of alarms and actions can support them, by suppressing alarms that predictably follow others [2], by predicting the next alarm [3], or by redesigning the operator interface [4]. But the mined sequences have to be understood first, and a list of several hundred overlapping sequences is no help in that. Bántay and Abonyi observed that the frequent sequences are related to each other in two ways that can be drawn as networks. They form a tree under extension, whose edges carry the confidence of the extension (section 2.2.2). And they are supported by overlapping sets of traces, so their co-occurrence defines a similarity. The first relation gives a weighted network and, through a transitive closure, a similarity; the second gives a similarity directly. Multidimensional scaling turns each similarity into a map [1].

4.2 Theory

4.2.1 The weighted extension network (WN)

Let $\mathcal{F} = \{\varphi_1, \dots, \varphi_P\}$ be a prefix-closed set of frequent sequences, for instance one mined with a gap constraint. The *weighted extension network* has a node for every φ_i and an edge $\varphi_i \to \varphi_j$ whenever $\varphi_i = \pi(\varphi_j)$, weighted by the confidence $c(\varphi_j) = \sigma(\varphi_j)/\sigma(\varphi_i)$. The network is a forest; each tree is rooted at a frequent single event, the *start event* of its sequences. In the drawing of the paper the color of a node encodes the length of the sequence, its size the cumulative confidence $c^*(\varphi_j)$, the share of the sequences of the start event that reach it, and the width of an edge the confidence [1]. Two layouts suit a forest: a *radial* layout that puts one start event in the centre and the longer sequences on concentric rings, and a *hierarchical* layout with one row per length. A tree can be pruned at a minimum confidence, which removes the weak continuations and everything below them.

The WN answers questions of the form *how does a process continue after this sequence, and how decisively?* A node with one thick outgoing edge is followed by a predictable step; a node with many thin edges is a branching point. Its limitation is that sequences of different trees, or far apart in the same tree, are not related in the picture at all.

4.2.2 The confidence-based map (CM)

The CM view extends the confidences of the extension tree to all pairs of sequences. Let $\mathbf{A} \in [0,1]^{P\times P}$ be the *confidence matrix*, $a_{ij} = c(\varphi_j)$ if $\varphi_i = \pi(\varphi_j)$ and $a_{ij} = 0$ otherwise, and let $\mathbf{S}^{(0)} = \max(\mathbf{A}, \mathbf{A}^\mathsf{T})$ be its symmetrization. Two sequences of the same tree are connected by a unique path, and the paper defines their similarity as the product of the confidences along it; sequences of different trees are not similar at all. The definition is the *max-product transitive closure* of $\mathbf{S}^{(0)}$,

$$a^*_{ij} = \max_{\text{paths } i=k_0,k_1,\dots,k_r=j} \prod_{q=1}^{r} s^{(0)}_{k_{q-1}k_q}, \qquad a^*_{ii} = 1, \tag{4.1}$$

which in a forest reduces to the product along the unique path. Because all weights lie in $[0,1]$, a product can only shrink along a path, and one pass of the Floyd–Warshall recursion $a^*_{ij} \leftarrow \max(a^*_{ij}, a^*_{ik}a^*_{kj})$ over all intermediate nodes k in the $(\max,\times)$ semiring computes the closure exactly. The dissimilarity $\mathbf{D} = \mathbf{1} - \mathbf{A}^*$ is then embedded in the plane by metric multidimensional scaling (MDS) [20]: coordinates $\mathbf{y}_i \in \mathbb{R}^2$ are sought that minimize the *stress*

$$\sigma_1(\mathbf{Y}) = \sqrt{\frac{\sum_{i<j}\big(\|\mathbf{y}_i - \mathbf{y}_j\| - d_{ij}\big)^2}{\sum_{i<j} d_{ij}^2}}, \tag{4.2}$$

the normalized stress-1 of Kruskal [77]. `spmviz` minimizes it by majorization (SMACOF), started from the deterministic classical (Torgerson) solution, and reports σ_1 with every map; values around 0.1 are good, values above 0.3 warn that the plane cannot hold the dissimilarities faithfully [20].

In the CM map the sequences of one tree form a cloud around their start event; a long, confident chain of extensions becomes a ridge reaching away from it, and trees with similar structure lie side by side. The view answers *which frequent sequences belong to the same scenario?*

4.2.3 The transaction-based map (TM)

The TM view ignores the extension tree and compares the *supports*. Let $\mathbf{T} \in \{0,1\}^{P\times N}$ be the pattern–sequence incidence matrix, $t_{ik} = 1$ if sequence k supports φ_i. The similarity of two patterns is the Jaccard index of their supporting sets,

$$s_{ij} = \frac{\left|T_i \cap T_j\right|}{\left|T_i \cup T_j\right|} = \frac{(\mathbf{T}\mathbf{T}^{\mathsf{T}})_{ij}}{\sigma(\varphi_i) + \sigma(\varphi_j) - (\mathbf{T}\mathbf{T}^{\mathsf{T}})_{ij}}, \tag{4.3}$$

where T_i is the set of sequences supporting φ_i; $\mathbf{D} = \mathbf{1} - \mathbf{S}$ is embedded by MDS as before [1]. Patterns that occur in the same traces — the events of a sub-process that runs interleaved with others — form clusters, whatever their start events. The view answers *which frequent sequences occur together?*, the question behind the segmentation of a log into sub-processes. The Dice, overlap and cosine coefficients are available as alternatives to the Jaccard index.

4.2.4 Fused similarities

The paper closes by asking how the similarities could be combined [1]. The CM and TM similarities see different things, and a third one is natural: the similarity of the patterns as strings, the length of their longest common subsequence divided by the longer length (`content_similarity`). `spmviz` combines them in two ways. A weighted mean is transparent but sensitive to the different scales of the similarities. *Similarity network fusion* (SNF), developed for combining patient similarities from several genomic data types [74], avoids the scale problem: each similarity is turned into a row-normalized matrix $\mathbf{P}^{(v)}$ and a sparse local kernel $\mathbf{K}^{(v)}$ that keeps the k nearest neighbours of every pattern, and the matrices are diffused through each other,

$$\mathbf{P}^{(v)} \leftarrow \mathbf{K}^{(v)}\Big(\frac{1}{V-1}\sum_{u\neq v}\mathbf{P}^{(u)}\Big)\mathbf{K}^{(v)\mathsf{T}}, \qquad v = 1,\dots,V, \tag{4.4}$$

for a fixed number of iterations; the fused similarity is the average of the $\mathbf{P}^{(v)}$, rescaled to $[0,1]$. Strong local agreements between the similarities are reinforced, weak disagreements fade. `NBVFS("fused")` uses SNF of the confidence, transaction and content similarities by default and a weighted mean when weights are given.

Algorithm 4.1 NBVFS: weighted network and maps

Input: prefix-closed pattern set $\mathcal{F}$ with supports and supporting sequences; view $\in$ {WN, CM, TM}
Output: a graph with node coordinates

build the extension network: an edge $\pi(\varphi_j) \to \varphi_j$ with weight $c(\varphi_j)$;
if *view* = *WN* **then**
    lay out the forest radially (one start event in the centre) or by length; prune edges below $c_{\min}$;
else
    if *view* = *CM* **then**
        $\mathbf{S} \leftarrow$ max-product closure of $\max(\mathbf{A}, \mathbf{A}^{\mathsf{T}})$, eq. (4.1);
    else
        $\mathbf{S} \leftarrow$ Jaccard similarity of the supporting sets, eq. (4.3);
    $\mathbf{Y} \leftarrow$ SMACOF on $\mathbf{D} = \mathbf{1} - \mathbf{S}$ from the classical solution; report the stress eq. (4.2);
attach support, length, confidence, cumulative confidence and start event to every node;

4.2.5 Communities and sub-logs

The second open question of the paper is how to segment the network [1]. A map invites the eye to find clusters, and an algorithm can find them as well: `pattern_communities` builds the k-nearest-neighbour graph of a similarity matrix and partitions it by Louvain modularity maximization [75]. With the transaction similarity, a community gathers patterns supported by the same traces, a candidate sub-process. The traces that support the patterns of a community form a *sub-log*, which can be mined, drawn as a process map or compared with the other sub-logs; `sublogs` returns one `SequenceDatabase` per community. The resolution parameter of the modularity sets the granularity.

4.3 Using it in spmviz

The `NBVFS` class follows the estimator convention of scikit-learn: the constructor takes the options, `fit(ps)` computes the network and the coordinates, and the fitted object exposes `graph_` (a NetworkX `DiGraph` with coordinates `x`, `y` and all pattern attributes on the nodes), `positions_`, `similarity_`, `stress_` and `patterns_`. `plot()` draws the view, `to_frame()` returns the node table and `write(path)` writes the graph to GEXF, GraphML, Cytoscape JSON or node-link JSON with positions and colors, ready for Gephi [78] or Cytoscape [79]. The options are the view (`"wn"`, `"cm"`, `"tm"`, `"fused"`), `start_items` (the filter of the paper), `min_confidence`, `layout` of the WN, `mds` (`"smacof"`,

In a nutshell Network-based visualization of frequent sequences

- **WN**: the extension forest of the frequent sequences, edges weighted by confidence, nodes sized by cumulative confidence — how sequences grow and where they branch.
- **CM**: max-product closure of the confidences, embedded by MDS — which sequences belong to the same scenario.
- **TM**: Jaccard similarity of the supporting traces, embedded by MDS — which sequences occur together.
- **Fused map**: similarity network fusion of the confidence, transaction and content similarities.
- **Communities**: Louvain communities of a similarity graph segment the pattern set and the log into sub-processes.
- Every map comes with its stress; every network is a NetworkX graph with all attributes.

`"classical"`, `"sklearn"`, or neighbourhood embeddings for very large sets), `weight` (`"confidence"` as defined in the paper, or `"cumulative"`, the edge value of the reference scripts [5]) and `similarity_weights` of the fused map. The building blocks are public: `extension_network`, `confidence_matrix`, `transitive_confidence`, `transaction_similarity`, `content_similarity`, `fuse_similarities`, `snf`, `mds`, `pattern_communities` and `sublogs`.

```
db = sv.datasets.load_clickstream()
ps = sv.mine(db, min_support=0.01, algorithm="cmspam", max_gap=1,
    max_length=10)
print(ps, "lengths", ps.lengths.min(), "-", ps.lengths.max())
print({k: db.label(k) for k in range(4)})
```

```
A = confidence_matrix(ps)              # a_ij = c(phi_j) if phi_i is the
    parent of phi_j
S = transitive_confidence(A)           # max-product closure of max(A, A^T)
print("non-zero similarities:", (A > 0).sum(), "->", (S > 0).sum() -
    len(ps))
```

4.4 Reproducing the paper

The paper analyses two logs, both bundled with the package. The *click-stream* contains 343 sessions over 14 services of the unemployment-benefit application of the Dutch Employee Insurance Agency, part of the BPI Challenge 2016 data [29]; the services are coded 0–13 in order of first appearance, as in the paper. The *alarm log* of a hydrofluoric-acid alkylation plant contains 28,536 alarms (codes 100–499, printed A…) and operator actions (codes above 1000, printed O…) in 3,956 traces recorded between September and December 2020. As in the paper,

CM-SPAM mines the click-stream with 1% minimum support, maximum gap 1 and maximum length 10, and the alarm log with 1% minimum support and maximum gap 2.

`spmviz` finds 241 frequent sequences of length 1–8 in the click-stream and 421 of length 1–6 in the alarm log; the paper reports 242 and 420 with the same length ranges [1]. The five algorithms of the package agree with each other and with brute-force counting on both logs, so the differences of one pattern must come from the preparation of the data. The reference scripts convert the logs to the SPMF format with a Java converter and concatenate the item codes into strings before computing the parents [5]; the click-stream records some clicks with minute resolution only, so the order of simultaneous clicks depends on how the file is read. The bundled click-stream keeps the order of the original file. Such details shift the support of a sequence at the threshold by one and are immaterial for the maps, but they illustrate a general point: a reproducible analysis needs the exact data preparation as much as the mining parameters.

4.5 Examples

The weighted network of the click-stream. The paper filters the network by the starting service; fig. 4.1 shows the sessions that start with service 2, Hours worked, in the radial layout. The hierarchical layout of fig. 4.2 draws the trees of four start events side by side, and fig. 4.3 prunes the weak extensions: at a minimum confidence of 0.4 only the decisive continuations remain.

```
wn = sv.NBVFS("wn", layout="circle", start_items=[2]).fit(ps)
wn.plot(labels=20, title="NBVFS-WN, sequences starting with 'Hours
    worked'")
```

The confidence map. The closure turns the 227 non-zero confidences of the 241 click-stream sequences into 7,372 non-zero similarities. Figure 4.4 highlights three start events; the normalized stress of 0.41 is high, as expected when 241 points are forced into a plane while sequences of different trees are all at distance one. Figure 4.5 compares classical scaling with SMACOF.

```
cm = sv.NBVFS("cm").fit(ps)
print("normalized stress:", round(cm.stress_, 3))
cm.plot(start_items=[2, 3, 4], labels=16)
```

The transaction map finds sub-processes. The synthetic alarm floods of `make_alarm_cascades` interleave cascades of four sub-processes — pump, cooling, compressor and separator — with nuisance alarms; some floods contain two cascades. The TM map of the frequent sequences (fig. 4.6) separates the four cascades without knowing them; fig. 4.7 highlights the cooling cascade, and fig. 4.8 shows how lower thresholds fill the space between the clusters with mixed combinations.

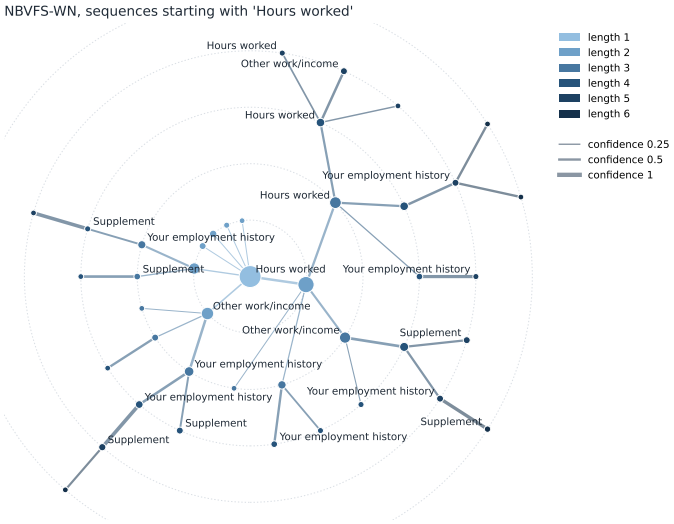


Figure 4.1. NBVFS-WN of the click-stream sessions starting with Hours worked: radial layout, node color = length, node size = cumulative confidence, edge width = confidence.

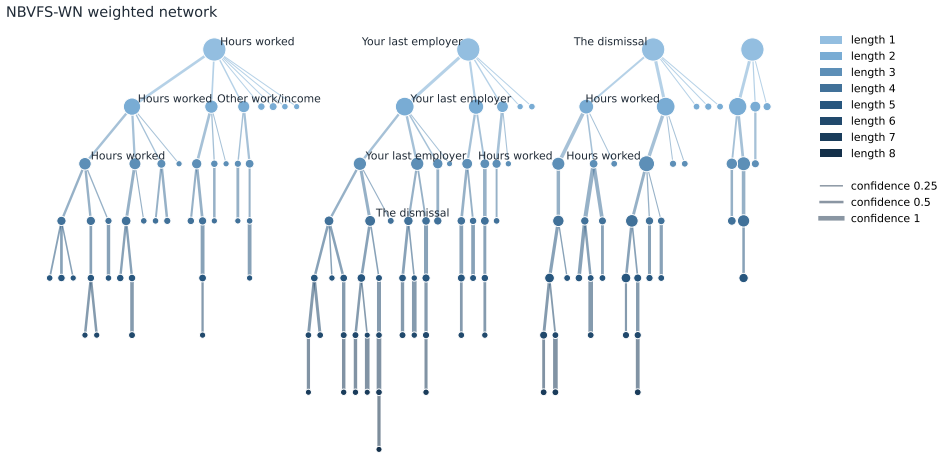


Figure 4.2. The extension trees of the first four services in the hierarchical layout, one row per length.

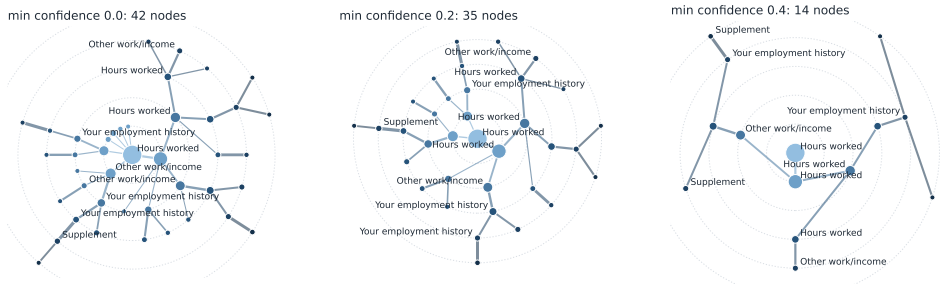


Figure 4.3. Pruning weak extensions of the Hours worked tree at minimum confidences 0, 0.2 and 0.4.

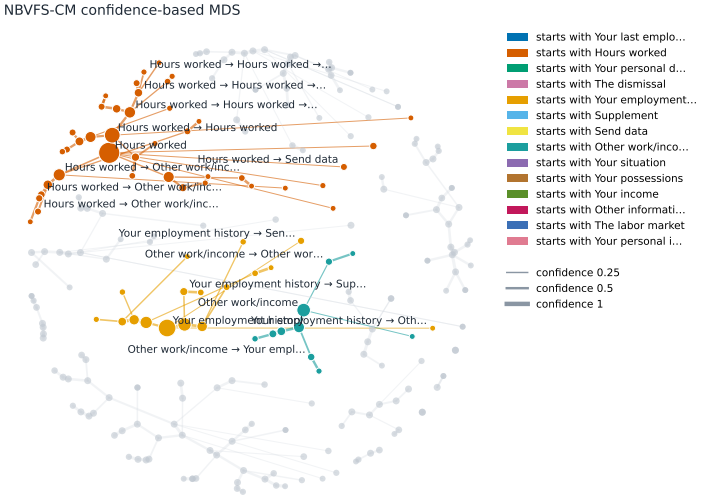


Figure 4.4. NBVFS-CM of the click-stream with three start events highlighted; edges join every sequence to its parent, node size is the support.

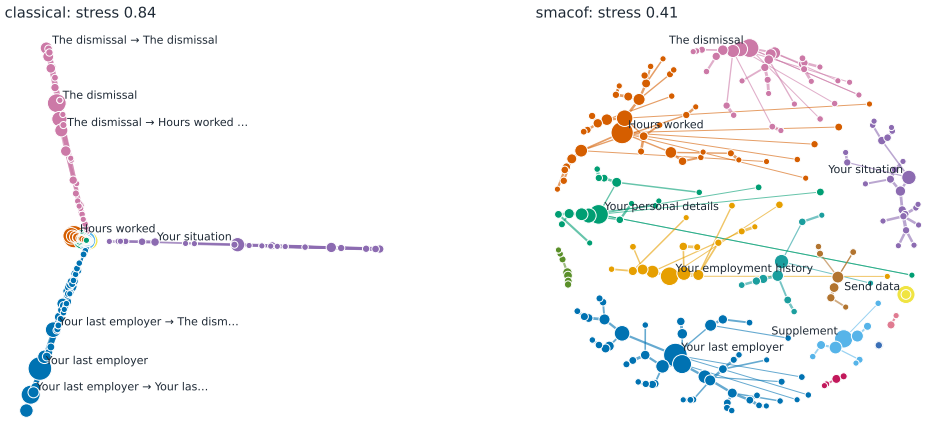


Figure 4.5. The CM map by classical scaling (left) and by SMACOF (right) with their stress values.

```
db = sv.datasets.make_alarm_cascades(600, random_state=0)
print(db.attributes["scenario"].value_counts().head())
ps = sv.mine(db, min_support=0.05, max_length=5)
tm = sv.NBVFS("tm").fit(ps)
tm.plot(labels=14)
```

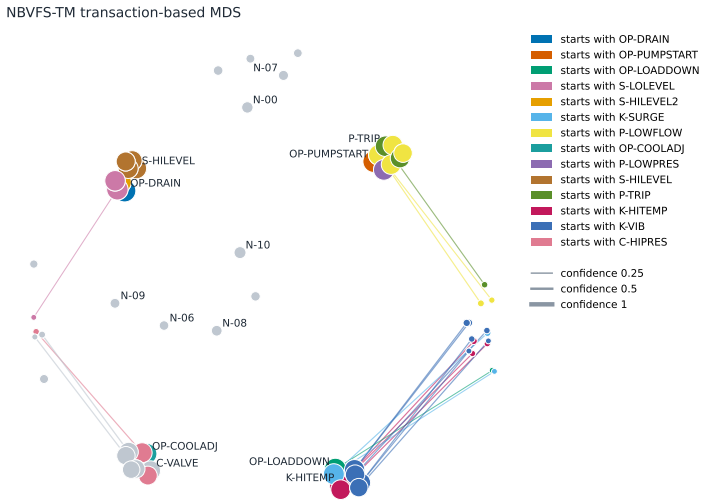


Figure 4.6. NBVFS-TM of synthetic alarm floods: the frequent sequences of the four interleaved sub-processes form four clusters.

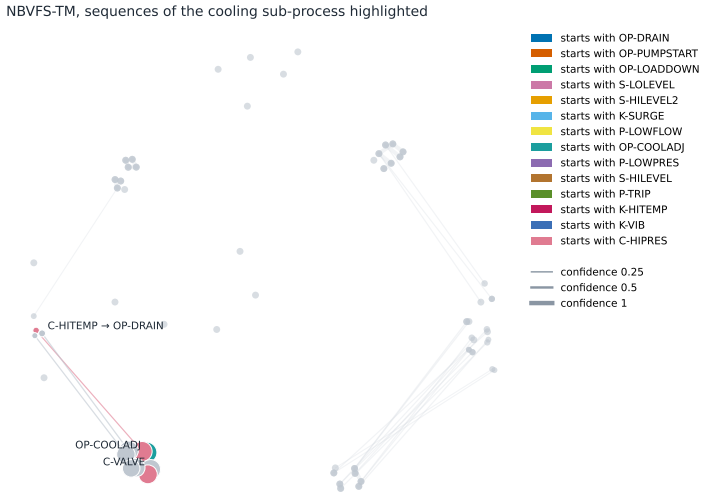


Figure 4.7. The same map with the sequences of the cooling sub-process highlighted.

The industrial alarm log. The most frequent events of the alkylation plant are operator actions (O1084, O1085, O1080, O1081) followed by the alarms A309 and A330. Figure 4.9 draws the extension tree of the most frequent start event;

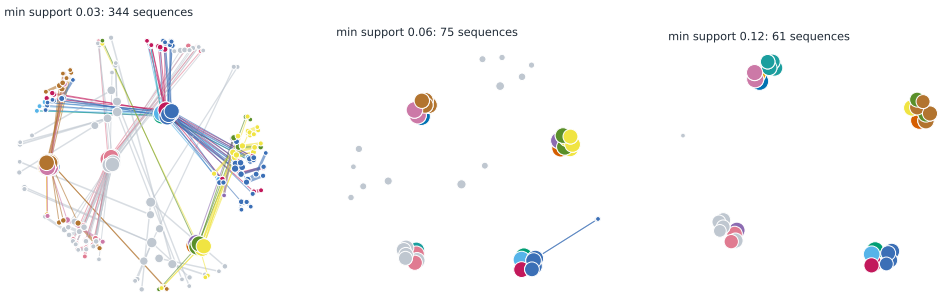


Figure 4.8. TM maps at minimum supports of 0.03, 0.06 and 0.12: rare combinations of cascades and nuisance alarms fill the space between the cores.

fig. 4.10 puts the CM and TM maps side by side. The repetitive sequences of operator actions, such as ⟨O1084 O1085 O1084 O1085 ···⟩, are long and confident chains, and the CM map stretches them along one dominant direction. The TM map shows no sharp clusters: the frequent operator actions occur in traces of many kinds, so the sequences of the three most frequent start events spread over the whole map.

```
db = sv.datasets.load_alarm_log()
ps = sv.mine(db, min_support=0.01, algorithm="cmspam", max_gap=2,
    max_length=10)
print(ps, "lengths", ps.lengths.min(), "-", ps.lengths.max())
top = ps.filter(min_length=1, max_length=1).sort("support")
print("most frequent events:", top.labels()[:8])
```

Beyond the paper. The fused map of the synthetic alarm floods (fig. 4.11) combines the three similarities. Louvain communities of the transaction similarity graph find five groups with modularity 0.78 (fig. 4.12): four of them coincide with the four sub-processes, the fifth gathers the floods in which the pump and the compressor cascades occur together. Each community defines a sub-log of 42 to 209 traces; fig. 4.13 draws one of them as a process map.

```
fused = sv.NBVFS("fused").fit(ps)
fused.plot(labels=14)
```

```
labels, modularity, G = pattern_communities(ps, similarity="transaction",
    knn=8)
print(f"{labels.max() + 1} communities, modularity {modularity:.2f}")
parts = sublogs(ps, labels)
for c, sub in parts.items():
    print(c, len(sub), "traces",
    sub.attributes["scenario"].value_counts().index[0])
```

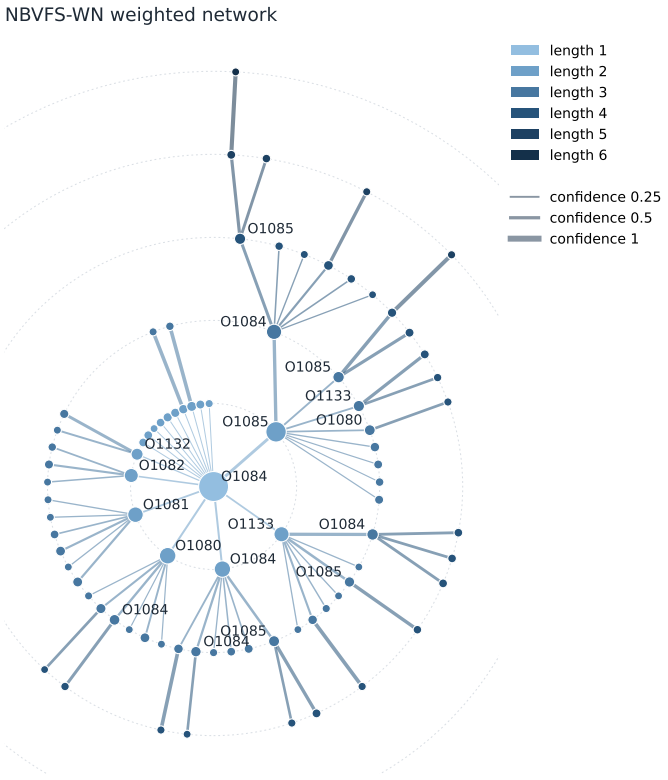


Figure 4.9. NBVFS-WN of the industrial alarm log: the extension tree of the most frequent start event, O1084.

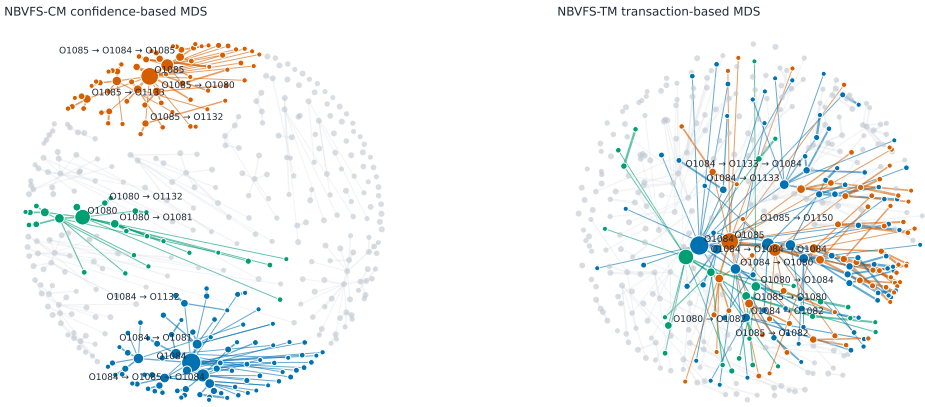


Figure 4.10. NBVFS-CM (left) and NBVFS-TM (right) of the 421 frequent sequences of the alarm log, the three most frequent start events highlighted.

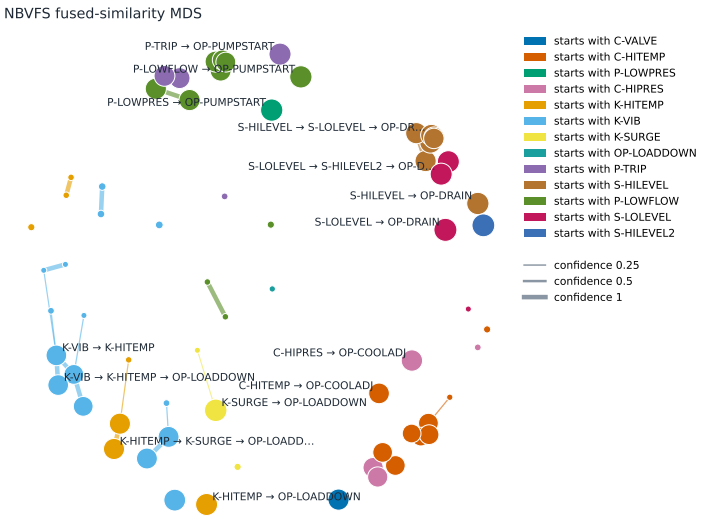


Figure 4.11. The fused map: similarity network fusion of the confidence, transaction and content similarities of the synthetic alarm floods.

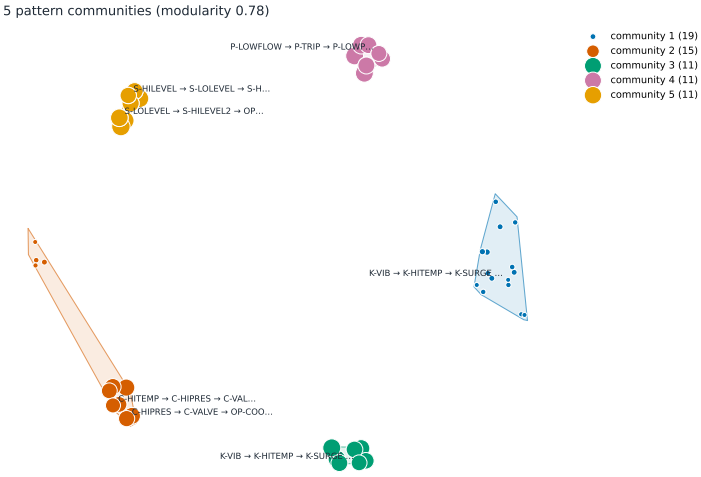


Figure 4.12. Louvain communities of the transaction similarity graph: candidate sub-processes.

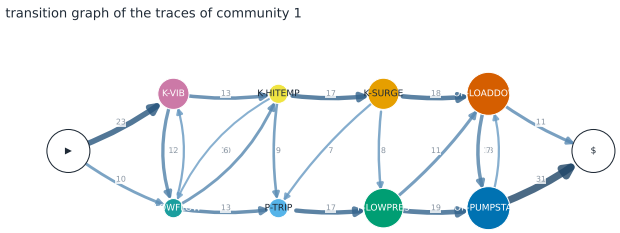


Figure 4.13. The sub-log of the first community, the floods in which the pump and the compressor cascades occur together, drawn as a directly-follows graph (chapter 6).

CHAPTER

5



Trees of prefixes and patterns

> Merging the common beginnings of the sequences gives a tree that every analyst of event data knows in some form. This chapter defines the prefix tree and the pattern tree, the anchored trees behind word trees, and the drawings of a tree as a node-link diagram, a sunburst, an icicle and a time-aware LifeFlow. It introduces two new colorings of the tree: by the entropy of the next step, which marks the points where a process branches, and by the contrast between two groups of cases.

5.1 Problem and idea

How do the cases begin? Which second step follows a given first step, and how often does a case end there? A tree answers these questions without any mining: every prefix that occurs in the data is a node, the number of cases passing through it is its weight, and the children of a node are the steps that follow it. The tree is the aggregation behind the earliest visual tools for web sessions [53], behind LifeFlow [54], EventFlow [55] and CoreFlow [56], and, grown from a chosen word, behind the word tree of text visualization [57]. Its weakness is that it only sees prefixes: the same three steps in the middle of two cases fall into different branches. Two remedies exist. The tree can be grown from an *anchor*, a pattern chosen by the analyst, in both directions (the double tree [58, 59]). Or the tree can be built from the frequent sequences instead of the raw prefixes: the extension tree of section 2.2.2 is a tree of patterns that may start anywhere in a case.

5.2 Theory

5.2.1 Prefix trees and pattern trees

The *prefix tree* (trie) of a database $\mathcal{D}$ has a root $\emptyset$ and a node v for every prefix $\langle s_1 \cdots s_d \rangle$ that occurs at the start of at least one sequence; v is a child of the node of $\langle s_1 \cdots s_{d-1} \rangle$. Its weight $n(v)$ is the number of sequences that start with the prefix, and $n_\$(v)$ the number of those that end there. The weights satisfy $n(v) = n_\$(v) + \sum_{u \in \text{children}(v)} n(u)$, the conservation law that makes the tree drawable as a space-filling diagram. A depth limit and a minimum weight keep the tree readable; `spmviz` also offers to keep only the k heaviest children of every node.

The *pattern tree* of a prefix-closed pattern set is the extension tree with the supports as weights. It differs from the prefix tree in two ways: a pattern may start anywhere in a sequence and skip events, and the weights do not satisfy the conservation law, because a sequence can support several children of the same node. Both trees are drawn by the same functions; which of the two is drawn depends on whether a database or a pattern set is passed.

An *anchored* tree starts every sequence at the first occurrence of an anchor pattern α. The *forward* tree is the prefix tree of the suffixes after the occurrence, the *backward* tree the prefix tree of the reversed prefixes before it; together they form the double tree of α [58]. Sequences that do not contain α are left out.

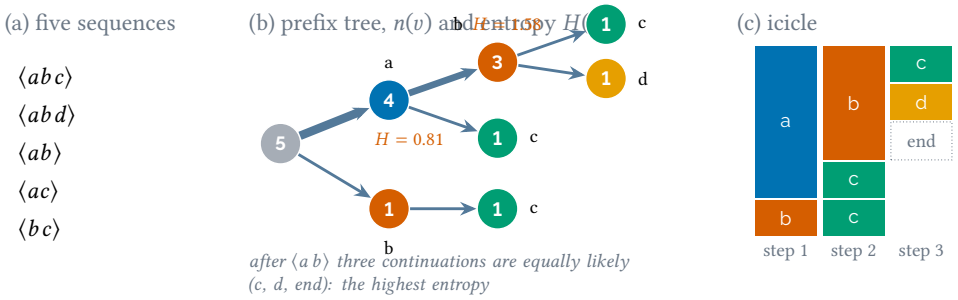


Figure 5.1. A prefix tree, its continuation entropies and its icicle. (b) Node weights are the numbers of sequences starting with the prefix; $H(v)$ is the entropy of the next step, the end of the sequence included. (c) The icicle stacks the nodes of each depth with heights proportional to their weights; the gap under a node is the share of sequences that end there. A sunburst is the same diagram in polar coordinates.

5.2.2 Drawing a tree

A *node-link* drawing places the depths on one axis and orders the nodes of each depth on the other, giving every leaf an equal share and every inner node the mean position of its children; edges are drawn as Bézier curves as wide as the weight of the child. It shows the topology best and labels well, but it wastes space on wide trees. Space-filling drawings give every node an area proportional to its

weight. The *icicle* stacks the nodes of each depth as rectangles in a column [52], the *sunburst* as wedges of a ring, angle proportional to weight [51]; the share of the cases that end at a node is the empty part below it (fig. 5.1). Sunbursts are memorable and compact but make angles hard to compare; icicles compare better and label better.

LifeFlow replaces the depth axis by time: every node is placed at the mean time elapsed from the start of the sequences to the event, so horizontal gaps show how long the steps take [54]. A *word tree* draws the forward tree of an anchor with font sizes proportional to the weights [57]; the *double tree* adds the backward tree on the left [58].

5.2.3 The continuation-entropy tree

Where is the next step of a process predictable, and where do the cases scatter? Let $P(e \mid v) = n(u_e)/n(v)$ be the share of the sequences through node v that continue with event e (the child u_e), and $P(\$ \mid v) = n_\$(v)/n(v)$ the share that end. The *continuation entropy* of the node is

$$H(v) = -\sum_{e \in \mathcal{I} \cup \{\$\}} P(e \mid v) \log_2 P(e \mid v) \quad \text{bits}, \tag{5.1}$$

zero when the next step is certain and $\log_2 K$ when K continuations are equally likely. Coloring the prefix tree by $H(v)$, from green (predictable) to vermilion (branching), makes the decision points of a process visible at a glance (fig. 5.1b). For monitoring and process improvement these are the places where a warning, a recommendation or a redesign pays off; for prediction they are the places where no model can be sure. The same quantity is defined for a pattern of a pattern set, with the supports of its extensions in place of the child weights (`measures.continuation_entropy`); comparing the mean entropy of patterns of increasing length tells whether a longer history makes the next step more predictable.

5.2.4 The contrast tree

When the cases fall into groups — mobile and desktop visitors, converted and lost customers, shifts of a plant — the tree can show where the groups differ. Let $a(v)$ and $b(v)$ be the numbers of sequences of the positive group (size N_1) and of the rest (size N_0) that pass through v. The contrast tree colors each node by the smoothed log-ratio of the relative frequencies,

$$r(v) = \log_2 \frac{(a(v) + \frac{1}{2})/(N_1 + 1)}{(b(v) + \frac{1}{2})/(N_0 + 1)}, \tag{5.2}$$

on a diverging scale: blue nodes are typical of the positive group, vermilion nodes of the rest, white nodes of neither. The pseudocounts keep r finite where one group

Algorithm 5.1 Anchored prefix tree with entropy and contrast

Input: database $\mathcal{D}$, optional anchor α and direction, depth limit d, minimum weight $n_{\min}$, optional groups
Output: a rooted tree with weights n, $n_\$$, entropies H and contrasts r

foreach *sequence* s **do**
  if *an anchor is given* **then**
    if $\alpha \not\sqsubseteq s$ **then** skip s;
    $s \leftarrow$ suffix after (forward) or reversed prefix before (backward) the first occurrence of α;
  walk down the tree along $s_1, \dots, s_{\min(d,|s|)}$, creating nodes and incrementing n; increment $n_\$$ at the last node if the sequence ends there;
remove nodes with $n(v) < n_{\min}$; optionally keep the k heaviest children of every node;
compute $H(v)$ by eq. (5.1); with groups, repeat the counting per group and compute $r(v)$ by eq. (5.2);

In a nutshell Trees

- The prefix tree aggregates the beginnings of the cases; the pattern tree draws the frequent sequences, which may start anywhere.
- Node-link trees show topology, icicles and sunbursts show proportions, LifeFlow shows durations, word and double trees show the context of an anchor.
- The continuation-entropy tree marks where the next step is unpredictable.
- The contrast tree marks where two groups of cases differ; the contrast table adds growth rates and Fisher's exact test.

never passes. For frequent sequences, `measures.contrast` adds the growth rate of Dong and Li's emerging patterns [76] and the p-value of Fisher's exact test for every pattern, so that the visual impression can be checked statistically.

5.3 Using it in spmviz

`networks.prefix_tree(db, max_depth, min_count, anchor, direction)` returns the tree as a NetworkX `DiGraph` with the attributes `item`, `depth`, `count`, `n_end`, `path` and, with timestamps, `mean_time`; `networks.double_tree` returns the two halves of a double tree. The drawing functions `plot_prefix_tree`, `plot_sunburst` and `plot_icicle` accept a database (prefix tree) or a pattern

set (pattern tree) and share the options `max_depth`, `min_count`, `max_children`, `anchor` and `color_by` (`"item"`, `"entropy"` or `"contrast"` with `groups` and `positive`); `plot_icicle(time_aware=True)` gives LifeFlow, `plot_double_tree(db, anchor)` the double tree. `interactive.sunburst_figure` and `icicle_figure` return zoomable plotly versions.

5.4 Examples

A spectacular sunburst. `load_chess_openings` draws two thousand games from a small opening book of 33 main lines; every game is the sequence of its first moves (plies). In the sunburst of fig. 5.2 the familiar families appear as large slices: the open games after 1.e4 e5, the Sicilian Defence after 1.e4 c5, the closed games after 1.d4 d5 and the Indian systems after 1.d4 Nf6. The icicle of fig. 5.3 shows the same tree with labels that can be read, and fig. 5.4 the effect of the depth limit. The contiguous frequent sequences of the games ($g_{\max} = 1$, 5% support) recover the main lines: ⟨e4 e5 Nf3 Nc6⟩ is played in 468 games, the Sicilian ⟨e4 c5 Nf3 d6 d4 cxd4⟩ in 156.

```
db = sv.datasets.load_chess_openings(2000, random_state=0)
plot_sunburst(db, max_depth=5, min_angle=7, title="the first five plies of
    2,000 games")
```

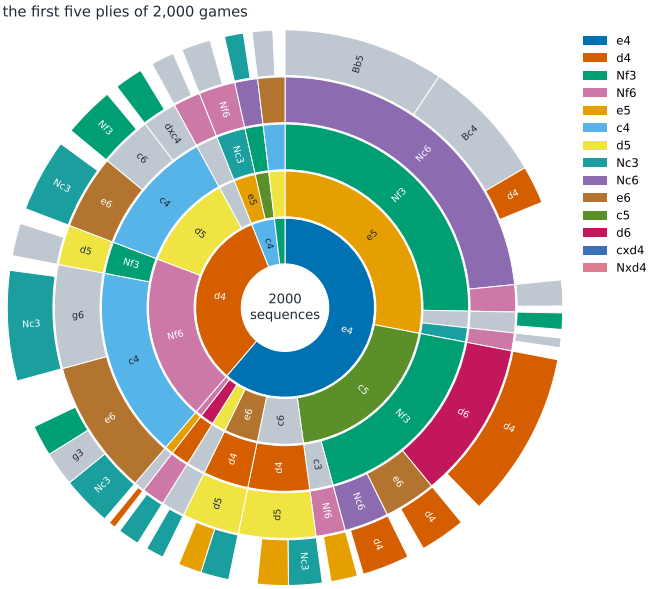


Figure 5.2. The first five plies of 2,000 games from an opening book: rings are plies, angles are numbers of games.

Where does the shop branch? The thousand synthetic journeys of a web shop start at the home page and continue to search, category and product pages, reviews,

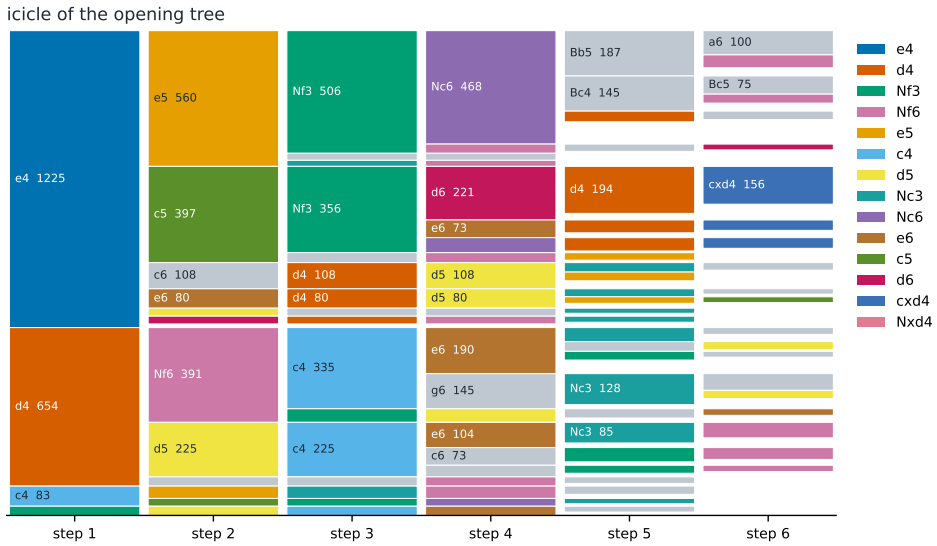


Figure 5.3. The opening tree as an icicle: columns are plies, heights are numbers of games, empty space marks games that leave the tree.

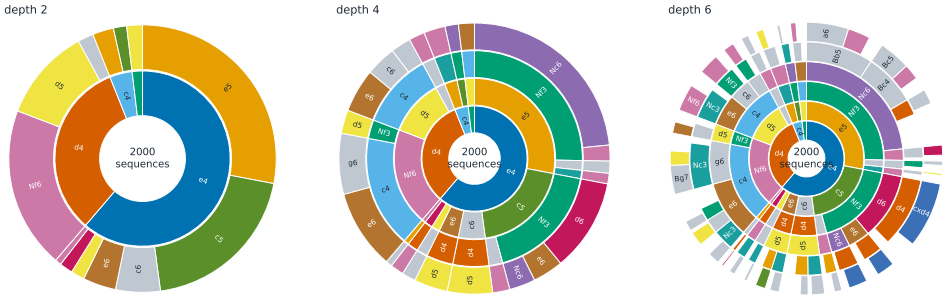


Figure 5.4. Sunbursts of the opening tree at depths 2, 4 and 6.

the cart, checkout and payment, or leave. In the continuation-entropy tree of fig. 5.5 the product pages are vermilion: after a product page visitors scatter almost evenly between the cart, leaving and reading reviews (after ⟨home category product⟩ the three shares are 0.29, 0.25 and 0.21). The mean entropy of the frequent sequences hardly falls with their length (1.08 bits for single pages, 0.94 bits for four pages): in this shop the history says little about the next page.

```
db = sv.datasets.make_customer_journeys(1000, random_state=0)
plot_prefix_tree(db, max_depth=4, min_count=25, color_by="entropy",
                 title="continuation entropy of the web-shop journeys")
```

Word trees. What happens before and after a visitor puts something into the cart? The double tree of fig. 5.6 answers in one picture. The same view works for

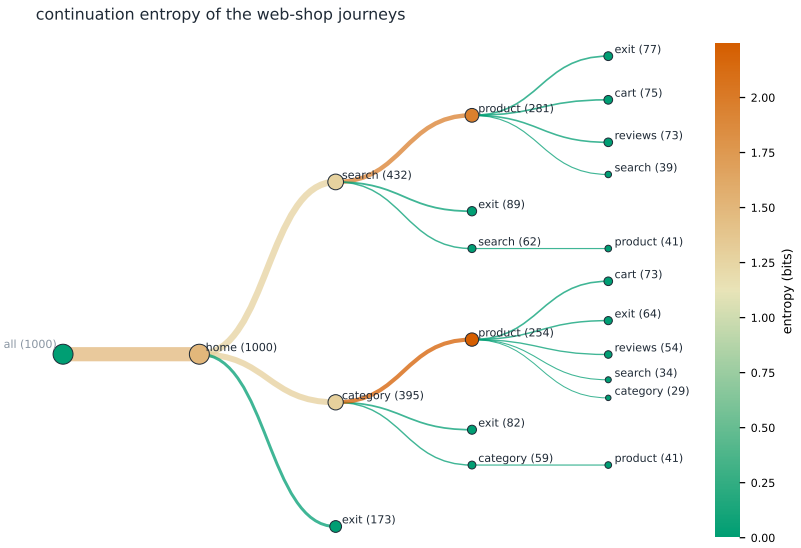


Figure 5.5. The continuation-entropy tree of a thousand web-shop journeys: green nodes have a predictable next page, vermilion nodes are decision points.

melodies: fig. 5.7 shows the context of the motif ⟨E D C⟩ in the phrases of thirteen public-domain songs, and fig. 5.8 the context of the longer pivot ⟨product cart⟩.

```
shop = sv.datasets.make_customer_journeys(1000, random_state=0)
plot_double_tree(shop, "cart", max_depth=3, min_count=15)
```

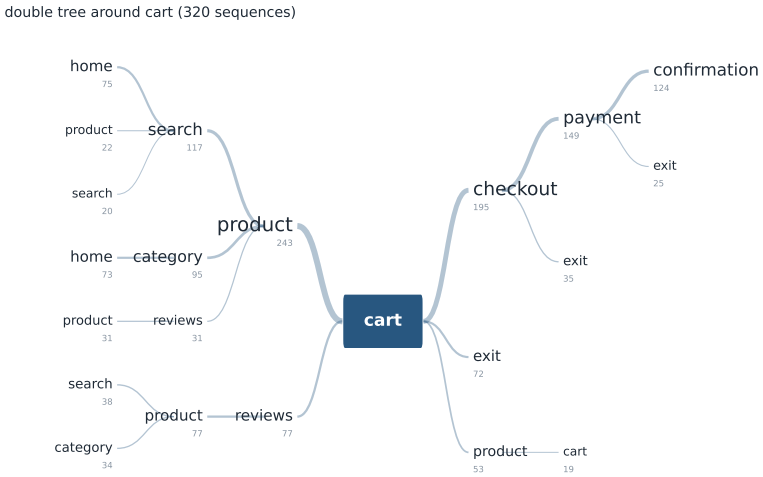


Figure 5.6. The double tree of the cart page: the steps before (left, read backwards) and after (right) the first visit to the cart.

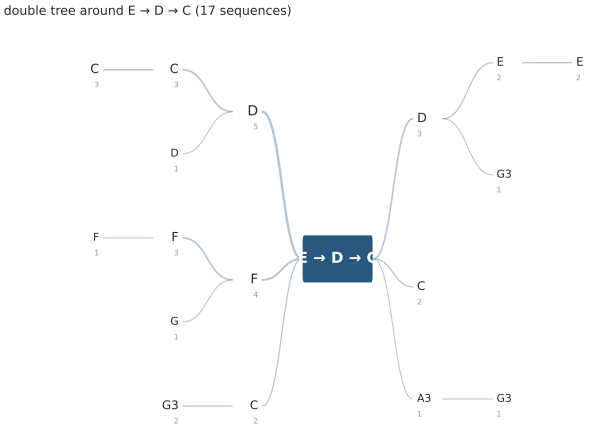


Figure 5.7. The double tree of the motif E–D–C in the phrases of thirteen songs.

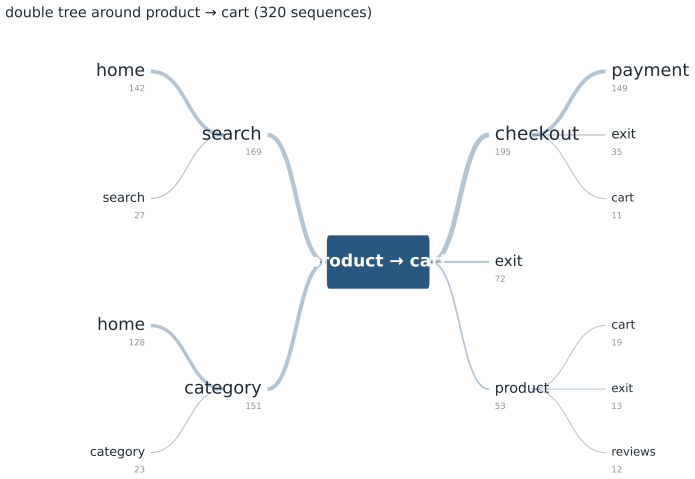


Figure 5.8. A longer pivot: the context of ⟨product cart⟩.

Contrast. Desktop visitors of the synthetic shop reach the cart, checkout and payment more often than mobile visitors. The contrast sunburst of fig. 5.9 shows where the paths diverge, and the contrast table confirms the impression: ⟨cart⟩ is supported by 39% of the desktop sessions, with a log-ratio of 0.65 against the mobile ones and a Fisher p-value of $4 \cdot 10^{-9}$. Figure 5.10 contrasts converted and lost visitors.

```
plot_sunburst(db, max_depth=4, color_by="contrast", groups="device",
    positive="desktop",
              title="desktop (blue) versus mobile (vermilion)")
ps = sv.mine(db, 0.03, max_gap=1, max_length=5)
table = sv.measures.contrast(ps, "device", positive="desktop")
print(table.head(8)[["pattern", "supp[desktop]", "supp[mobile]",
    "log2_ratio", "p_value"]])
```

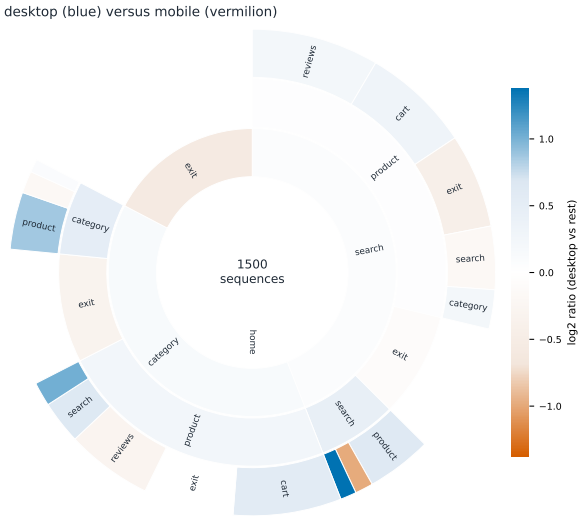


Figure 5.9. The contrast sunburst of desktop (blue) and mobile (vermilion) visitors.

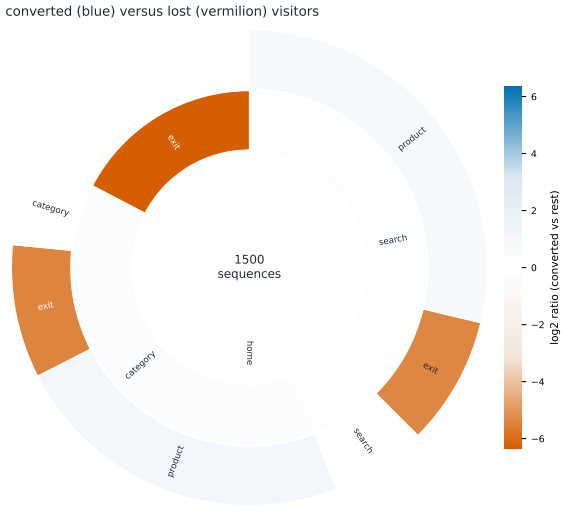


Figure 5.10. The contrast sunburst of converted (blue) and lost (vermilion) visitors, first three steps.

Time. The time-aware icicle of the daily routines (fig. 5.11) places every activity at the mean time since waking up. Split by persona (fig. 5.12), it shows the three mornings that the generator encodes: remote workers start work about half an hour after waking up, commuters reach work only after two hours, one of them on the road, and early birds go for a run before their coffee.

```
plot_icicle(db, max_depth=5, min_count=15, time_aware=True, title="mean
    time since waking up (hours)")
```

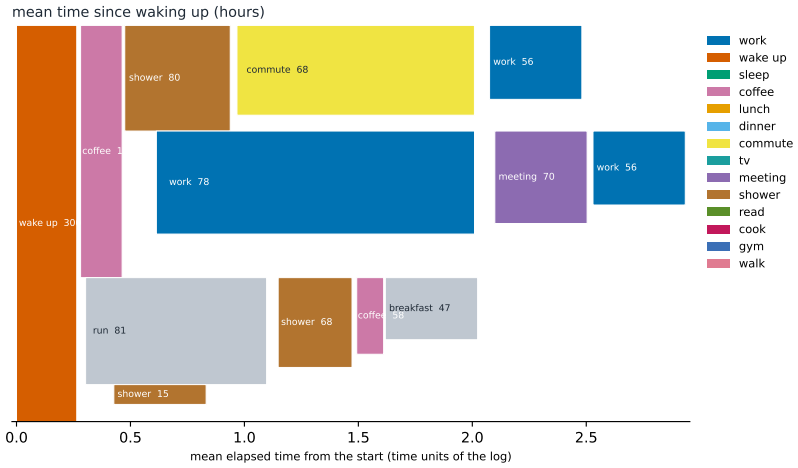


Figure 5.11. LifeFlow of the daily routines: every activity at the mean time since waking up (hours).

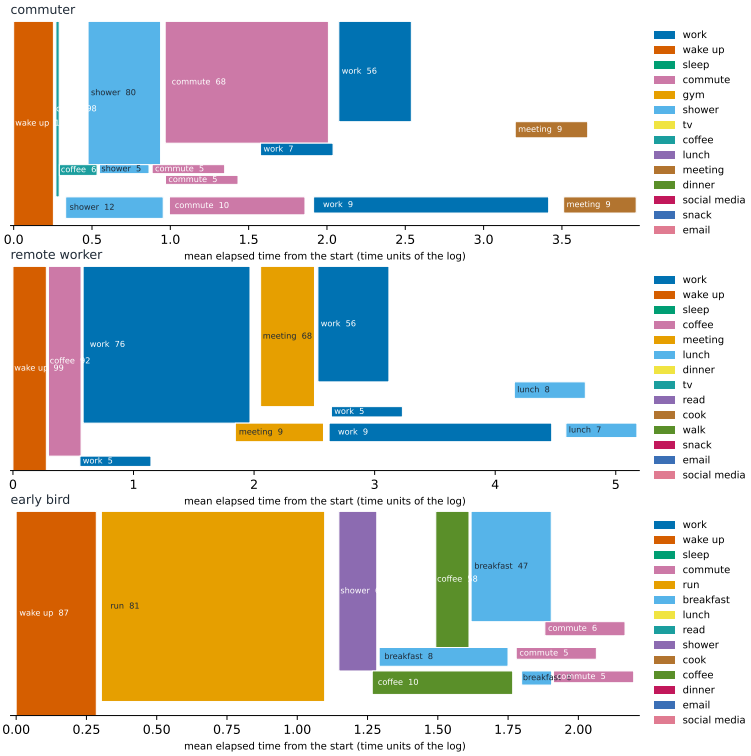


Figure 5.12. LifeFlow per persona: commuters, remote workers and early birds.

CHAPTER

6



Flows and transitions

> Flow views follow the cases from step to step. This chapter covers the Sankey diagram, the directly-follows graph and its matrix forms, MatrixWave and the chord diagram, all of which aggregate consecutive steps. It then introduces two views that connect flows with frequent sequences: the growth Sankey, which shows how support splits when frequent sequences grow, and the pattern-induced higher-order network, which keeps the history of a step only where frequent sequences show that the history matters.

6.1 Problem and idea

Trees keep the whole history of every prefix, and their size grows with it. Flow views make the opposite choice: they keep only what happens between two consecutive steps, at a given position (Sankey diagrams, MatrixWave) or anywhere in the sequences (directly-follows graphs, chord diagrams). The aggregation makes them compact and familiar — the directly-follows graph is the standard first view of process mining [17] — but it also makes them forget: a directly-follows graph cannot tell whether visitors who reach checkout came from the cart or from a saved basket, because both paths share the same edges. Frequent sequences carry exactly this missing information, and the two new views of the chapter use them to put the history back where it matters.

6.2 Theory

6.2.1 Sankey diagrams

A Sankey or alluvial diagram has one column per step $t = 1, \dots, T$. Column t stacks the events found at position t of the sequences, with heights proportional to their counts; a ribbon from event a in column t to event b in column $t + 1$ carries the sequences that have a at position t and b at position $t + 1$, and a terminal box collects the sequences that end at step t [45]. Frequence used the diagram for frequent event sequences of patients [46], and DecisionFlow made it scale to thousands of event types by merging [47]. To stay readable, each column keeps its k most frequent events and merges the others into other. With an *anchor* pattern α, the positions are counted from the end of the first occurrence of α, so the diagram shows what happens after α.

6.2.2 Directly-follows graphs and transition matrices

The *directly-follows graph* (DFG) has the events as nodes and an edge $a \to b$ weighted by the number of times b directly follows a anywhere in the sequences; artificial start and end nodes mark the first and last events. Normalizing every row gives the transition probabilities $P(b \mid a)$ of a first-order Markov chain, and the same numbers drawn as a heat map form the *transition matrix*. Node-link drawings of dense DFGs become unreadable; filtering the edges by frequency is the usual remedy, and matrices are the better choice when many events are connected [73]. MatrixWave [43] splits the transition matrix by position: one matrix for the transitions from step 1 to step 2, one from step 2 to step 3, and so on, arranged in a chain. A *chord diagram* arranges the events on a circle and draws a ribbon from the arc of a to the arc of b for every transition [44]; it shows loops and reciprocal flows well.

6.2.3 The growth Sankey

The extension tree of section 2.2.2 can be drawn as a flow as well. In the *growth Sankey*, column k holds frequent sequences of length k, a ribbon from $\pi(\varphi)$ to φ carries $\sigma(\varphi)$, and the rest of the support of a parent, $\sigma(\pi(\varphi)) - \sum \sigma(\varphi)$ over its drawn children, flows to a stop box. Each column keeps the k most supported children of every drawn parent. The diagram reads like a funnel of frequent sequences: it shows at a glance which continuations keep most of the support and how quickly the support of a start event thins out with length. The flow is exact when the children exclude each other, as in a prefix tree. When gaps are allowed, a case can continue a pattern in several ways and support several children, and the ribbons leaving a node can then carry more than the node itself — a visible reminder that frequent sequences, unlike prefixes, overlap.

Algorithm 6.1 Pattern-induced higher-order network

Input: database $\mathcal{D}$, maximum order K, minimum context support $\sigma_{\min}$
Output: a directed graph with first-order and memory nodes

count, for every contiguous context h of order $\leq K$, the events that follow it;
keep every single event as a first-order node;
foreach *kept context* h' *and every one-event-older context* h *that ends with* h' **do**
    if $\sigma(h) \geq \sigma_{\min}$ *and eq.* (6.1) *holds* **then**
        keep h as a memory node and examine its own older contexts against h
    else examine the older contexts of h against h';
foreach *kept context* h *and event* e *that follows it* **do**
    add an edge from the node of h to the node of the longest kept suffix of $h\,e$

6.2.4 Pattern-induced higher-order networks

A first-order network assumes that the next step depends on the current step only. Rosvall and co-authors showed that this assumption distorts the flows and communities of many real networks, and that second-order models correct them [49]. Xu, Wickramarathne and Chawla went further and let each node keep as much history as the data justify [50]. Their BuildHON algorithm compares, for every context $h = \langle e_{i-k+1} \cdots e_i \rangle$ of order k, the distribution of the next event $P(\cdot \mid h)$ with the distribution $P(\cdot \mid h')$ of the shorter context h' that drops the oldest event, and keeps h as a separate node when the Kullback–Leibler divergence exceeds a threshold that falls with the support of the context,

$$D_{\mathrm{KL}}\big(P(\cdot \mid h) \,\|\, P(\cdot \mid h')\big) = \sum_{e} P(e \mid h) \log_2 \frac{P(e \mid h)}{P(e \mid h')} \;>\; \frac{k}{\log_2\big(1 + \sigma(h)\big)}. \tag{6.1}$$

A context is examined only if its support reaches a minimum, which makes the contexts contiguous frequent sequences; the kept contexts are those frequent sequences whose continuation cannot be told from their last event alone. They become *memory nodes*: work|commute is the event work reached after commute. Every edge of the network leads to the node of the longest kept context of its target, so a walk on the network remembers exactly as much history as the frequent sequences require. We call the result a *pattern-induced* higher-order network. It is the flow-view counterpart of the continuation-entropy tree of section 5.2.3: the entropy tree marks where the next step is uncertain, the higher-order network where the history changes it.

In a nutshell Flows and transitions

- Sankey diagrams follow the cases step by step; an anchor shows what happens after a pattern.
- The directly-follows graph, the transition matrix, MatrixWave and the chord diagram aggregate consecutive pairs; they are compact but forget the history.
- The growth Sankey draws how the support of frequent sequences splits as they grow.
- The pattern-induced higher-order network keeps memory nodes exactly where frequent sequences change the next step significantly.

6.3 Using it in spmviz

`plot_sankey(db, max_steps, top, anchor)` draws the Sankey diagram, `plot_transition_graph`, `plot_transition_matrix` and `plot_matrixwave` the first-order views, `plot_chord` the chord diagram and `plot_growth_sankey(ps)` the growth Sankey. `networks.transition_graph(db)` returns the DFG as a NetworkX `DiGraph` with `count`, `probability` and `n_sequences` on the edges; `networks.higher_order_network(db, max_order, min_support, threshold)` returns the higher-order network with the attributes `item`, `history`, `order`, `count` and `entropy` on the nodes, and `plot_hon` draws it. `interactive.sankey_figure` gives an interactive Sankey diagram.

6.4 Examples

Sankey diagrams of a web shop. Figure 6.1 follows the thousand synthetic journeys of chapter 5 through five steps; fig. 6.2 anchors the diagram at the first visit to the cart and shows the four steps after it, and fig. 6.3 compares three and eight pages per column.

```
plot_sankey(db, max_steps=5, top=6)
```

Directly-follows graph, matrix and MatrixWave. Figure 6.4 is the DFG of the shop with its 24 strongest transitions; the edge from product to cart counts 273 transitions in 254 journeys, 29% of all transitions out of product pages. Figure 6.5 shows the full transition matrix, fig. 6.6 the MatrixWave of the first three steps, and fig. 6.7 the effect of the frequency filter.

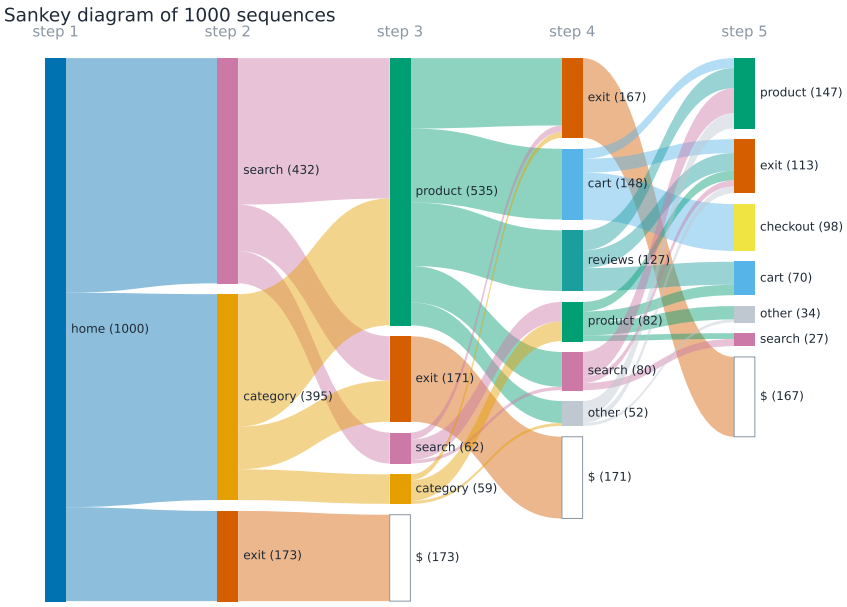


Figure 6.1. Sankey diagram of a thousand web-shop journeys, six pages per step; white boxes collect the visitors who leave.

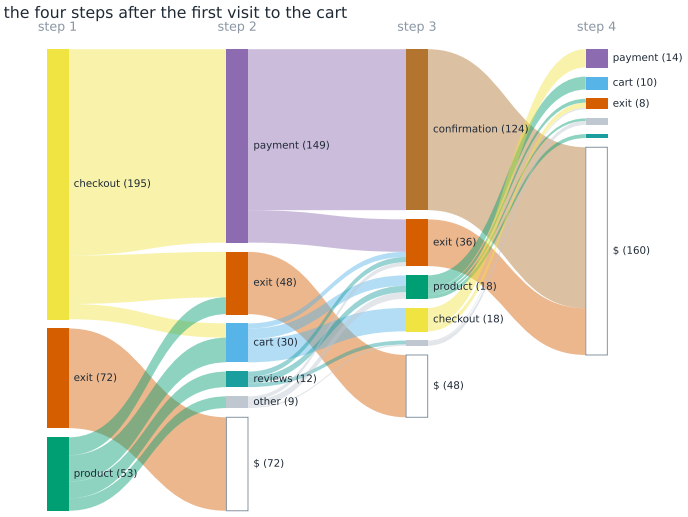


Figure 6.2. The four steps after the first visit to the cart (anchored Sankey diagram).

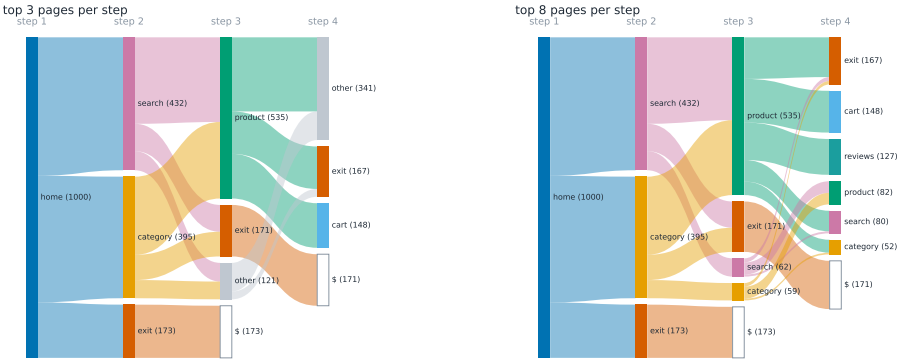


Figure 6.3. Three and eight pages per step: the merged other band shrinks, the diagram grows busier.

```
plot_transition_graph(db, top_edges=24)
G = sv.networks.transition_graph(db)          # networkx.DiGraph with
    counts and probabilities
print(G["product"]["cart"])
```

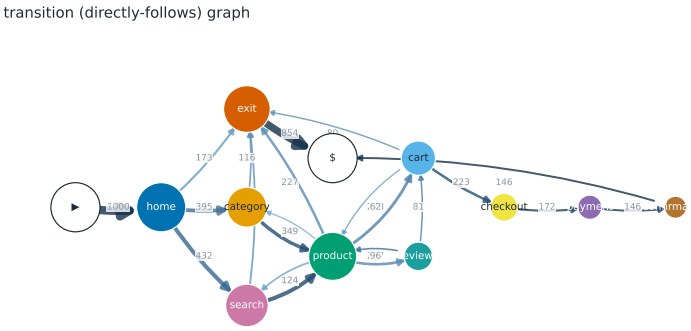


Figure 6.4. The directly-follows graph of the web shop with its 24 strongest transitions; the two white nodes mark the start and the end of the journeys.

Chord diagrams of a day. The chord diagram of the daily routines (fig. 6.8) shows the loops of a day at a glance; one chord per persona (fig. 6.9) shows how differently the three kinds of people move between their activities.

```
db = sv.datasets.make_daily_routines(300, random_state=0)
plot_chord(db, top=12)
```

Growth Sankey. Figure 6.10 draws how the support of the frequent sequences that start with wake up splits as they grow, three children per node; fig. 6.11 does the same for work and lunch.

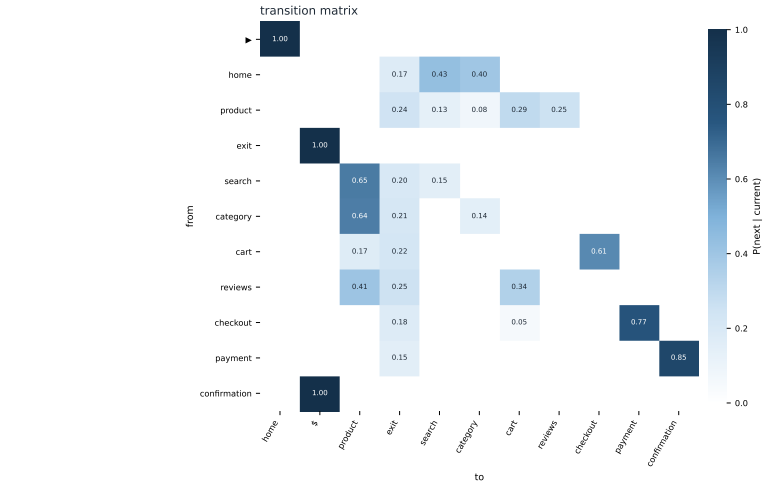


Figure 6.5. The row-normalized transition matrix of the web shop.

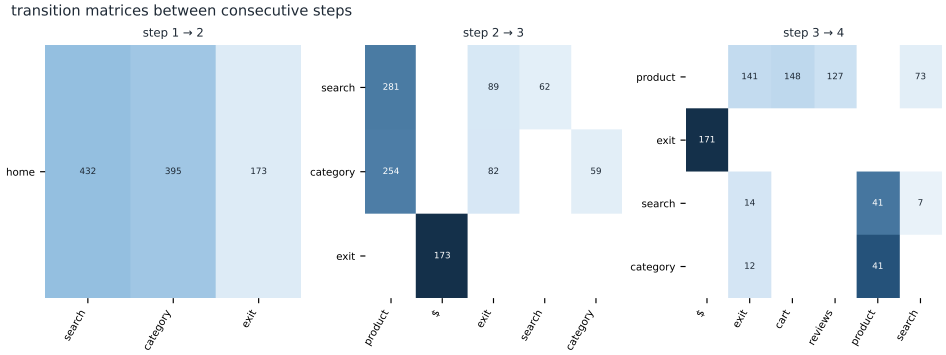


Figure 6.6. MatrixWave: one transition matrix per step, for the first three steps.

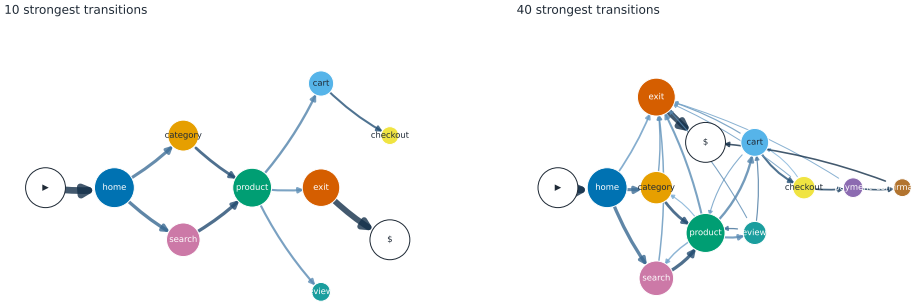


Figure 6.7. The ten and the forty strongest transitions.

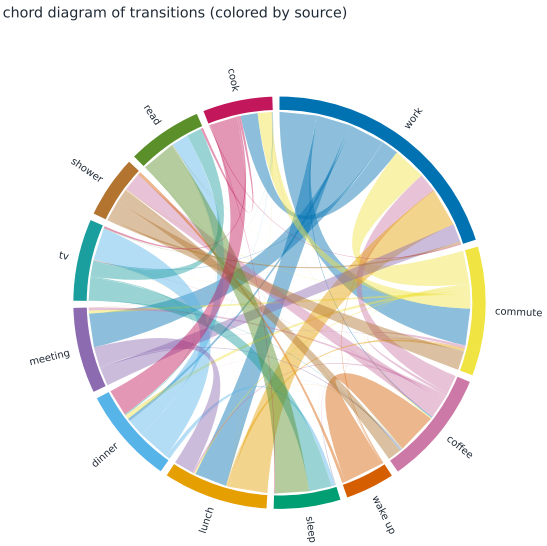


Figure 6.8. The chord diagram of the transitions of three hundred days, ribbons colored by their source activity.

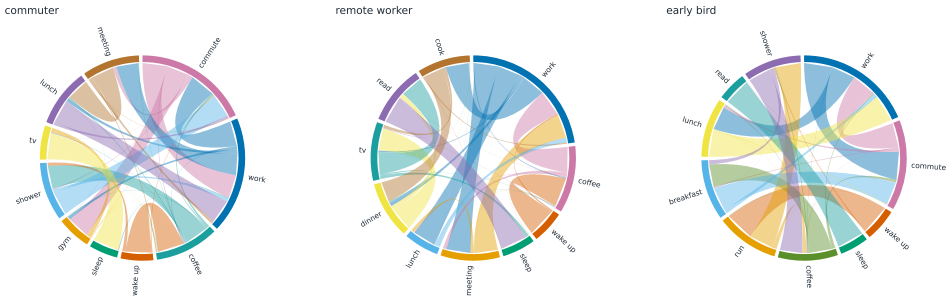


Figure 6.9. One chord diagram per persona: commuters, remote workers and early birds.

```
plot_growth_sankey(ps, start_items=["wake up"], max_length=5, top=3)
```

When history matters. In the daily routines the next activity after work depends on how work was reached. After the morning commute, 43% of the transitions go to lunch and 43% to a meeting; after lunch, 80% go to the commute home. The higher-order network of fig. 6.12 keeps 30 memory nodes, among them work|commute and work|lunch. The web-shop journeys are generated from first-order transition probabilities (one set for mobile and one for desktop visitors), and no memory node survives the test (fig. 6.13). The threshold of eq. (6.1) adapts to the support; fixed thresholds of 0.1, 0.5 and 1 bit keep 52, 30 and 18 memory nodes in the daily routines.

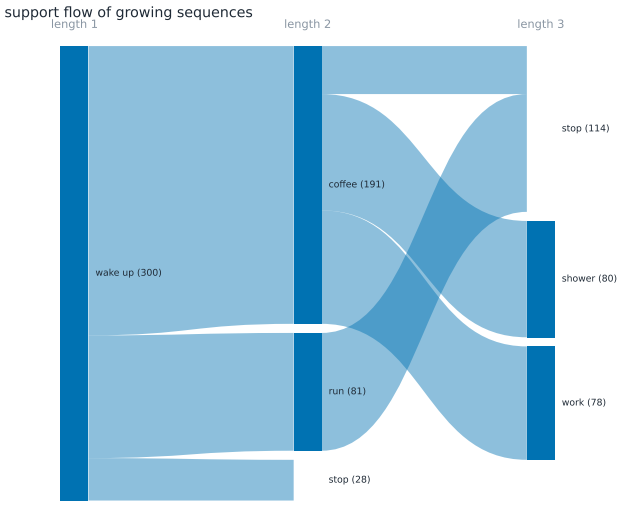


Figure 6.10. The growth Sankey of the frequent sequences that start with wake up: columns are lengths, ribbons carry supports, stop collects the support not continued by a drawn child.

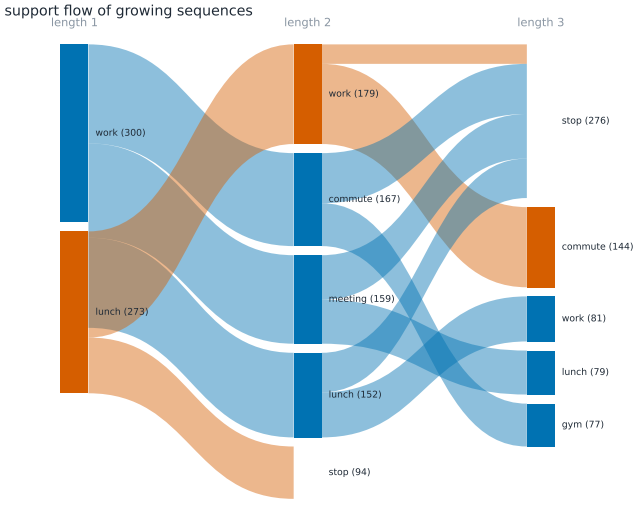


Figure 6.11. The growth Sankey of two start activities, work and lunch.

```
H = higher_order_network(db, max_order=3, min_support=10)
memory = [v for v, d in H.nodes(data=True) if d["order"] > 1]
print(len(memory), "memory nodes:", memory[:8])
for v in ("work|commute", "work|lunch"):
    if v in H:
        print(v, "->", {w: round(d["probability"], 2) for _, w, d in
    H.out_edges(v, data=True)})
plot_hon(H)
```

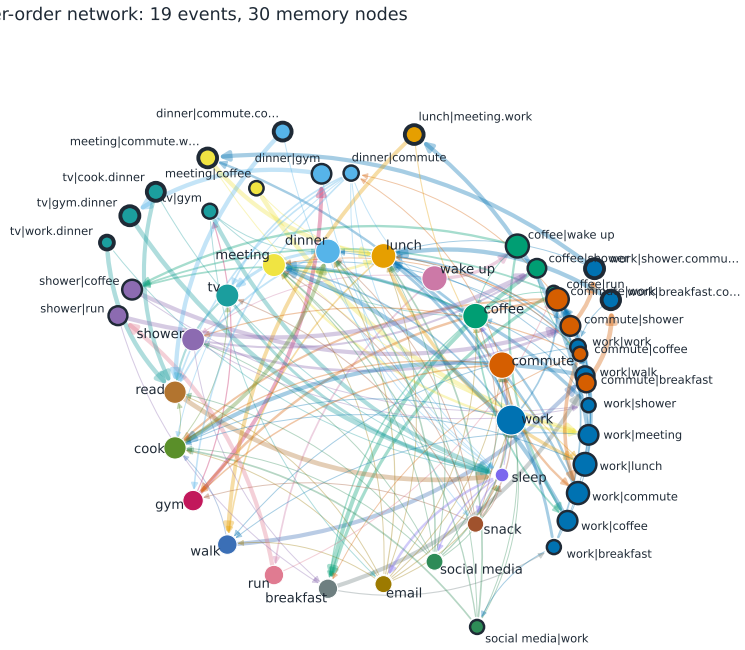


Figure 6.12. The pattern-induced higher-order network of the daily routines: 19 activities and 30 memory nodes (ringed), such as work|commute.

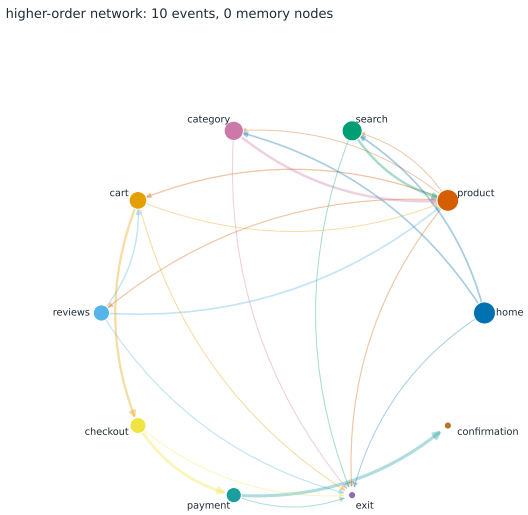


Figure 6.13. The higher-order network of the web-shop journeys: the first-order generator leaves no memory node.

CHAPTER

7



Matrices, maps and sets

> A pattern set has more structure than its extension tree: patterns are supported by overlapping sets of cases, they can be split into rules, they are ordered by containment, and they can be scored by interestingness measures. This chapter draws these structures as matrices (incidence, similarity, grouped rules), as points in the plane of two measures, as a terrain of pattern families, as a lattice, and as intersections of supporting sets. The pattern landscape is new.

7.1 Problem and idea

The network-based maps of chapter 4 compress the similarities of the patterns into two coordinates, and the compression costs accuracy. The matrices behind them can be drawn directly: the pattern–sequence incidence matrix, a pattern–pattern similarity matrix, or a matrix of rules. Matrix views do not overlap and scale to dense relations [73], but their message depends entirely on the order of the rows and columns, the problem of *seriation* [72]. Other questions call for other pictures. Which patterns are interesting rather than merely frequent is a question about measures, best answered by a scatter plot [64]; which patterns contain which is a question about an order, answered by a Hasse diagram; which patterns occur in the same cases is a question about sets, answered by an UpSet plot [67].

7.2 Theory

7.2.1 Incidence and similarity matrices

The incidence matrix $\mathbf{T} \in \{0,1\}^{P \times N}$ of section 4.2.3 has a row per pattern and a column per sequence. Its rows are ordered by the leaf order of a hierarchical clustering of the patterns (average linkage on $\mathbf{1} - \mathbf{S}$ with the Jaccard similarity $\mathbf{S}$, with optimal leaf ordering), its columns by the first pattern that each sequence supports. Blocks of the seriated matrix are groups of patterns supported by the same sequences; a row that spans several blocks is a pattern shared by several groups. The square similarity matrix $\mathbf{S}$, seriated in the same way and shown with its dendrogram, displays the same groups without the sequences. Any similarity of the book can be used: transaction (Jaccard of supports), confidence (eq. (4.1)), content (longest common subsequence) or a fused one.

7.2.2 Sequential rules and the grouped matrix

Splitting a pattern $\varphi = \langle \varphi_1 \cdots \varphi_m \rangle$ between two of its elements gives a *sequential rule* $\alpha \Rightarrow \beta$ with antecedent $\alpha = \langle \varphi_1 \cdots \varphi_q \rangle$ and consequent $\beta = \langle \varphi_{q+1} \cdots \varphi_m \rangle$. Its confidence and lift are

$$c(\alpha \Rightarrow \beta) = \frac{\sigma(\varphi)}{\sigma(\alpha)}, \qquad \operatorname{lift}(\alpha \Rightarrow \beta) = \frac{c(\alpha \Rightarrow \beta)\, N}{\sigma(\beta)} \ : \tag{7.1}$$

the share of the cases containing α in which β follows, and the ratio of this share to the frequency of β alone. Rules of this kind, read from frequent sequences, have been used for sequence prediction [37] and for alarm suppression [2]. The grouped matrix of Hahsler and Karpienko puts antecedents in the columns and consequents in the rows of a balloon plot, balloon area proportional to the support and color to a measure [65, 66]; `spmviz` orders both axes by the total support of their rules, so that the strongest rules gather in the upper left corner.

7.2.3 Measure scatter plots

Every pattern is a point in the plane of two measures — relative support and confidence, confidence and lift, support and length — and a thin line joins each pattern to its parent, so that the extension tree is still visible [64]. Frequent but unsurprising patterns sit at the right, rare but confident ones at the top left; lift separates real dependencies from coincidences of frequent events. Color can show the start event, the length or a group contrast.

7.2.4 Pattern landscapes

Maps of several hundred patterns are crowded, and the eye looks for regions rather than points. The *pattern landscape* turns a map into a terrain. Given coordinates

In a nutshell Matrices, maps and sets

- Seriated incidence and similarity matrices show groups of patterns supported by the same cases.
- The grouped rule matrix draws antecedents against consequents, with support and lift or confidence.
- Measure scatter plots separate interesting patterns from merely frequent ones.
- Pattern landscapes turn a map into a terrain of pattern families.
- The lattice shows the containment order, the UpSet plot the overlaps of the supporting sets.

$\mathbf{y}_i$ of the patterns, from the transaction map for instance, it draws the support-weighted Gaussian kernel density

$$f(\mathbf{y}) = \frac{\sum_i \sigma(\varphi_i)\, K_h(\mathbf{y} - \mathbf{y}_i)}{\sum_i \sigma(\varphi_i)}, \tag{7.2}$$

as contour levels, with the bandwidth h chosen by Scott's rule or by the analyst. Families of frequent patterns that occur in the same cases become mountains, isolated combinations remain in the plains, and every peak is named after the most supported long pattern near it. The landscape inherits the distortions of the map below it but makes its clusters and their weight visible at once.

7.2.5 Lattices and set views

Containment orders the frequent sequences. Their *Hasse diagram* has an edge $\varphi \to \psi$ whenever ψ has one item more than φ and contains it, wherever the item is inserted; drawn with one row per length and the closed patterns ringed, it shows how the pattern set is built from short patterns and where the support stops changing. The lattice is denser than the extension tree, in which every pattern has a single parent, so it is drawn for a few levels and the most supported patterns of each.

The supporting sets T_i of the patterns can also be compared as sets. The *UpSet* plot [67] lists the intersections of the supporting sets of a few patterns as rows of a dot matrix, with a bar for the number of sequences in each exact intersection and a bar for the support of each pattern. It answers a question that no map answers exactly: in how many cases do these patterns occur *together*, and in how many alone?

7.3 Using it in spmviz

`plot_incidence(ps)` and `plot_similarity_matrix(ps, similarity)` draw the seriated matrices, `seriate(S)` returns the order; `ps.incidence()` is the sparse matrix **T** and `networks.similarity_matrix(ps, name)` any of the similarities. `ps.rules(min_confidence)` (or `sequential_rules`) returns the rules as a data frame with confidence and lift, and `plot_rules_matrix(rules)` draws the grouped matrix. `measures.pattern_measures(ps)` computes support, confidence, cumulative confidence, lift, leverage, number of children and, optionally, entropy and timing for every pattern, and `plot_pattern_scatter(ps, x, y)` draws any two of them. `plot_landscape(ps, similarity)` draws the landscape (`embedding.landscape` returns the density grid); `plot_lattice` and `plot_upset` draw the set views, and `networks.pattern_lattice` returns the Hasse diagram as a graph.

7.4 Examples

Matrices of the alarm floods. Figure 7.1 is the seriated incidence matrix of the 40 most supported frequent sequences of the synthetic alarm floods of section 4.5: the four cascades appear as four blocks of rows supported by largely disjoint sets of floods; the floods with two cascades link the blocks. The seriated transaction similarity (fig. 7.2) shows the same blocks; the content similarity (fig. 7.3) groups patterns by their events rather than by their cases.

```
plot_incidence(ps, top=40)
plot_similarity_matrix(ps, "transaction", top=40)
```

Rules. The rules of confidence at least 0.6 of the alarm floods (fig. 7.4) pair every alarm with the operator action that ends its cascade: P-LOWFLOW ⇒ OP-PUMPSTART has confidence 1 and lift 2.99, K-SURGE ⇒ OP-LOADDOWN confidence 1 and lift 3.0. Figure 7.5 colors the same balloons by confidence.

```
rules = ps.rules(min_confidence=0.6)
print(rules.head(8)[["rule", "support", "confidence", "lift"]])
plot_rules_matrix(rules, top_antecedents=14, top_consequents=10)
```

Interesting patterns of the click-stream. Figure 7.6 places the 241 frequent sequences of the click-stream in the plane of relative support and confidence; fig. 7.7 in the plane of confidence and lift. The patterns with the highest lift are repetitions of Your situation and the pair ⟨The labor market Other information⟩: rare services that, once visited, are visited again or followed by a specific next one.

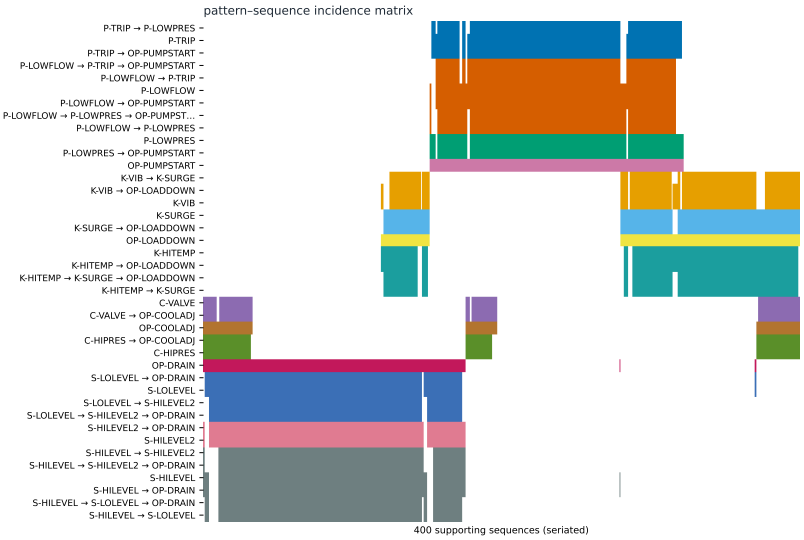


Figure 7.1. Seriated pattern–sequence incidence matrix of the synthetic alarm floods: four blocks, four cascades.

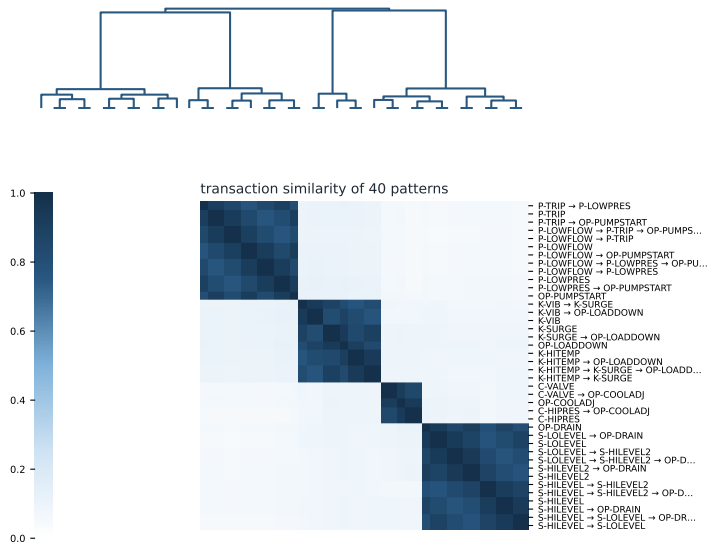


Figure 7.2. The seriated transaction similarity of the 40 patterns with its dendrogram.

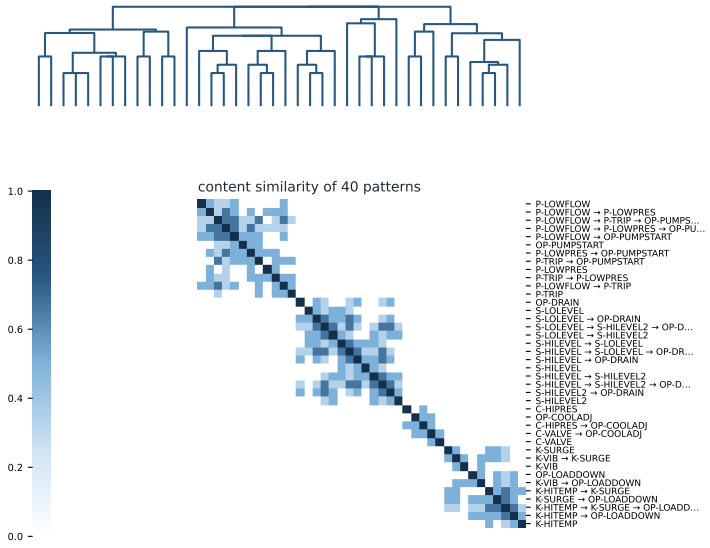


Figure 7.3. The seriated content similarity (longest common subsequence) of the same patterns.

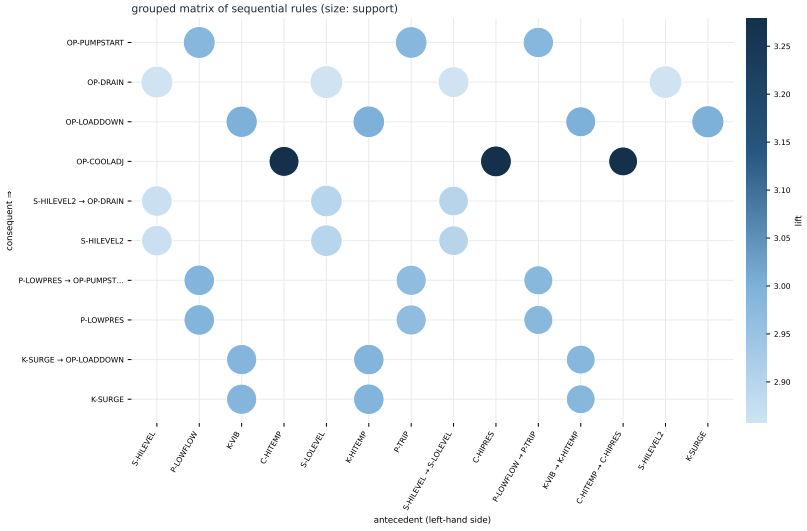


Figure 7.4. Grouped matrix of the sequential rules of the alarm floods: antecedents in columns, consequents in rows, balloon area = support, color = lift.

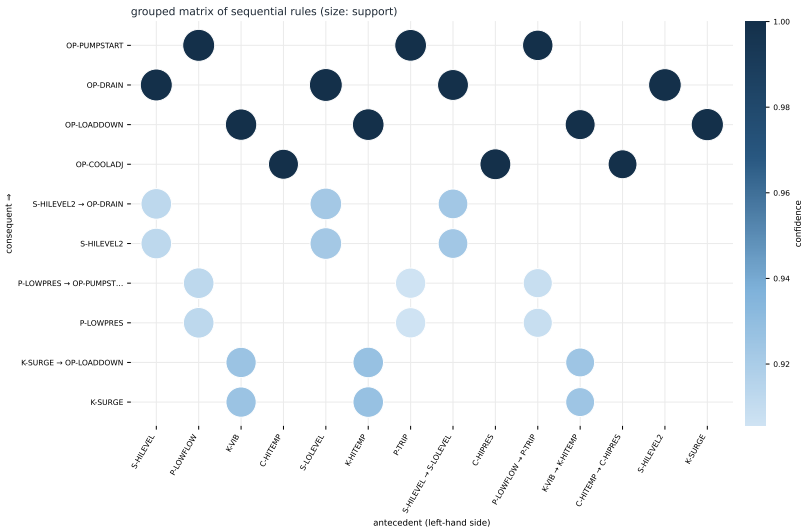


Figure 7.5. The same rules colored by confidence.

```
plot_pattern_scatter(ps, "rel_support", "confidence", labels=10)
plot_pattern_scatter(ps.filter(min_length=2), "confidence", "lift",
    color_by="start", labels=8)
```

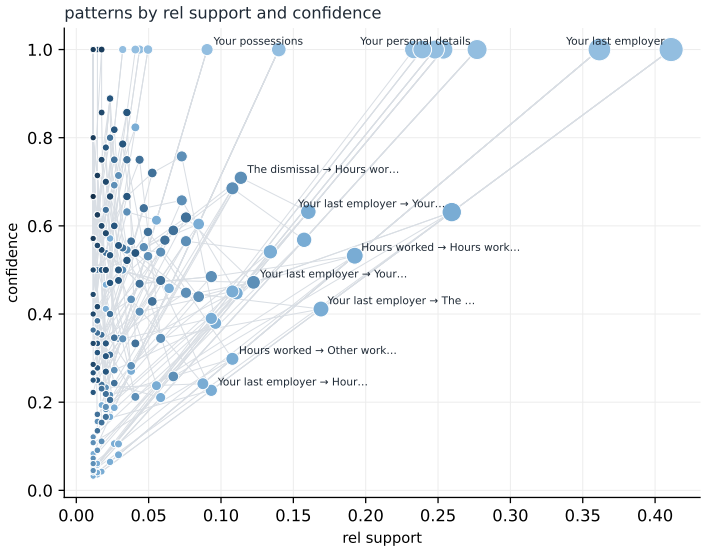


Figure 7.6. The click-stream patterns by relative support and confidence; grey lines join every pattern to its parent.

A landscape. The transaction map of the alarm floods becomes the landscape of fig. 7.8: four mountains named after their longest patterns, one per cascade. Figure 7.9 varies the bandwidth.

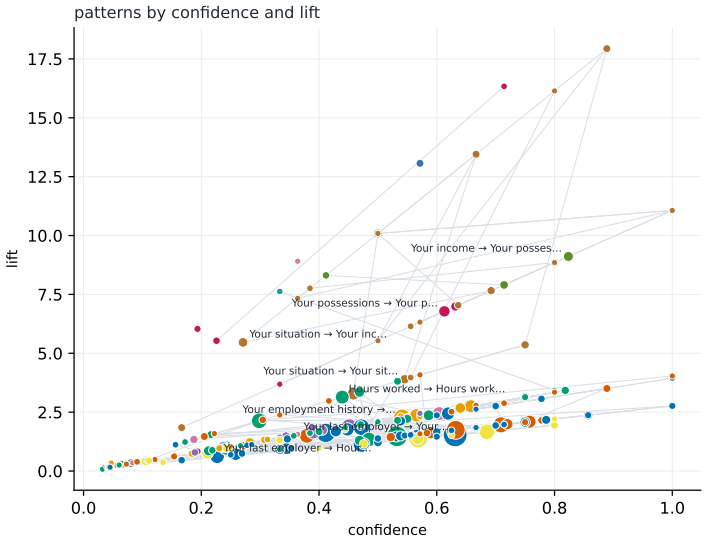


Figure 7.7. The click-stream patterns of length at least two by confidence and lift, colored by start event.

```
plot_landscape(ps, similarity="transaction", peaks=5)
```

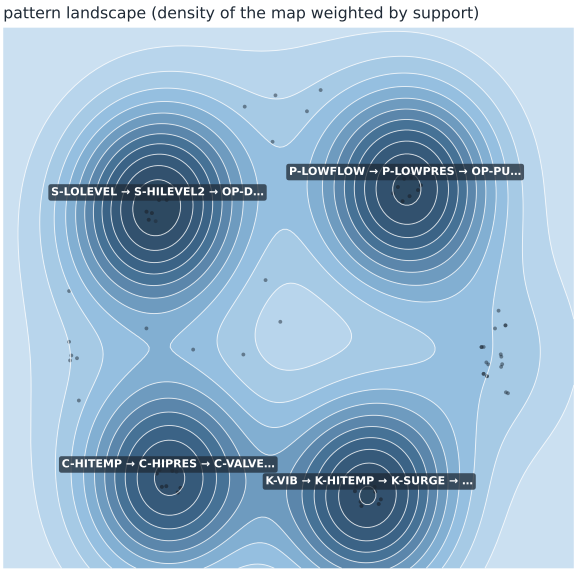


Figure 7.8. The pattern landscape of the synthetic alarm floods: support-weighted density of the transaction map, peaks named after their most supported long patterns.

Lattice and UpSet. Figure 7.10 is the containment lattice of the daily-routine patterns up to length four, nine patterns per level, closed patterns ringed. The

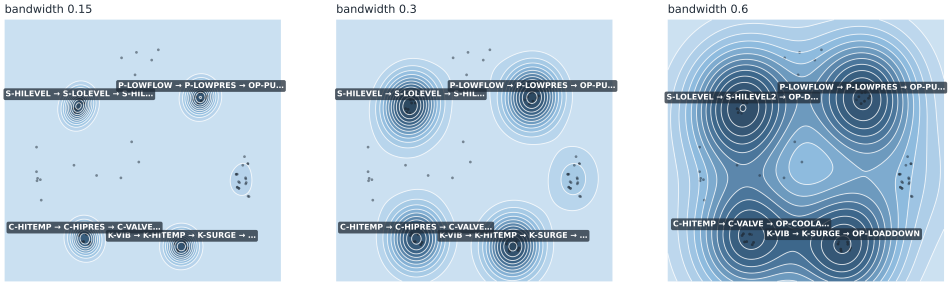


Figure 7.9. Landscapes with bandwidths 0.15, 0.3 and 0.6.

UpSet plot of the seven most supported patterns of length at least two (fig. 7.11) shows how often they occur together in the same day.

```
plot_lattice(ps, max_length=4, top_per_level=9)
plot_upset(ps.filter(min_length=2), top=7)
```

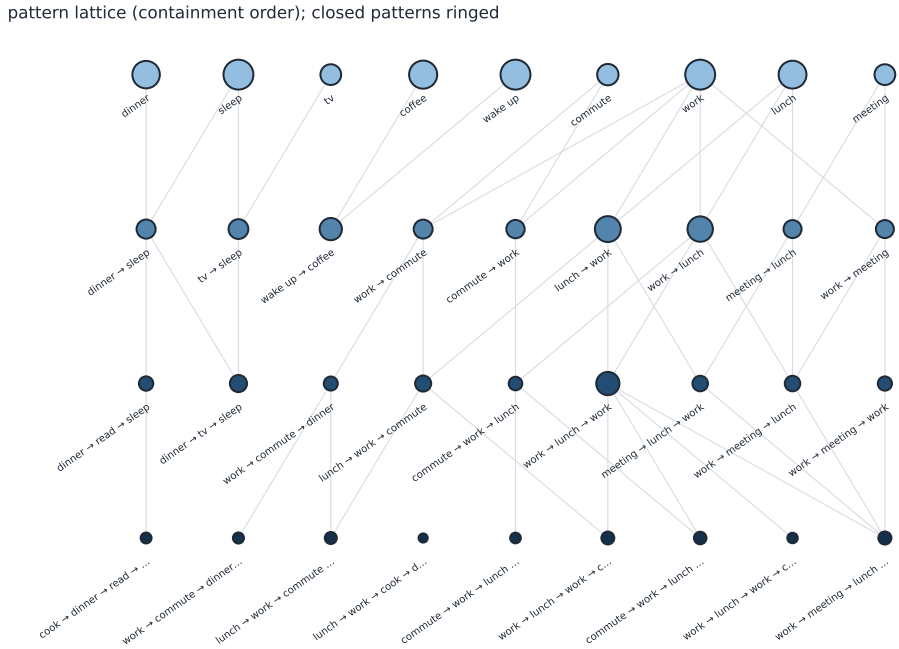


Figure 7.10. The containment lattice of the daily-routine patterns: one row per length, edges to one-item super-patterns, closed patterns ringed.

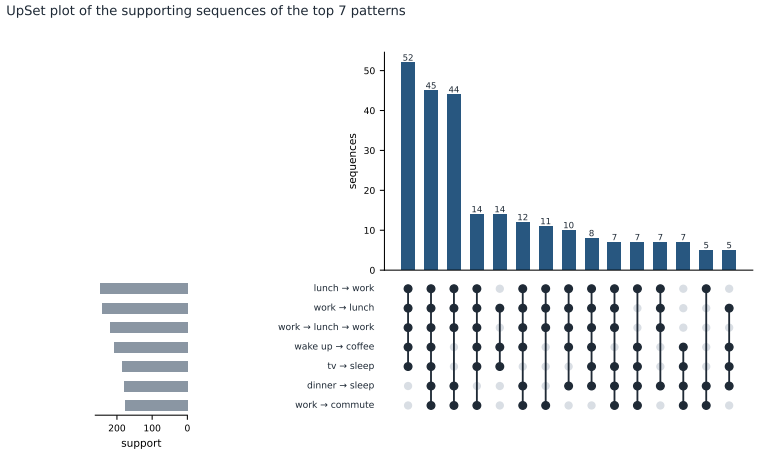


Figure 7.11. UpSet plot of the supporting days of the seven most supported patterns of length at least two.

CHAPTER

8



Sequence views and summaries

> After the trees, flows and maps of aggregated data, this chapter returns to the sequences themselves. It presents the views of social-science sequence analysis (index plots, chronograms, sequence frequency plots), the dotted chart of process mining, the sequence logo of bioinformatics and the arc diagram of text visualization, each combined with the mined patterns. It then treats two kinds of summaries: episodes, the frequent patterns of one long stream without case boundaries, and pattern sets selected by the minimum description length principle.

8.1 Problem and idea

Every aggregation hides something, and a pattern found by mining is only trustworthy once it has been seen in the data. The views of this chapter keep every case visible, or nearly so, and use the patterns to organize the picture: to sort the cases, to mark the occurrences of a pattern, or to align the cases at an anchor. They come from several traditions. Social-science sequence analysis, with its R package TraMineR, developed index plots, chronograms and frequency plots for life courses and careers [18, 19, 39]. Process mining contributed the dotted chart [40], bioinformatics the sequence logo [41], and text visualization the arc diagram [42]. The second half of the chapter deals with two situations in which the usual pattern set is the wrong summary: a log without case boundaries, where episodes replace sequential patterns [38], and a pattern set too large to read, where a summary by the minimum description length principle replaces it [68, 69].

8.2 Theory

8.2.1 Index plots, chronograms and frequency plots

A *sequence index plot* draws sequence i as row i of colored cells, one cell per element [19]. What it shows depends on the order of the rows: sorting by length shows the distribution of lengths, sorting by the first events shows the beginnings, and sorting by a pattern puts the supporting sequences first. `spmviz` adds the patterns to the picture: the earliest occurrence of each highlighted pattern is marked in every sequence that contains it, so the plot answers where a frequent sequence occurs and what surrounds it.

The *state distribution plot* or chronogram stacks, at every position t, the shares of the events among the sequences that are still running at t [18]; a line adds the cross-sectional entropy $-\sum_e p_t(e) \log_2 p_t(e)$, which is low where the cases are synchronized and high where they diverge. The *sequence frequency plot* draws the most frequent complete sequences as stacked rows with heights proportional to their frequency [18]. It is the right view when whole sequences repeat, as in short web sessions or standardized procedures, and an uninformative one when every case is unique.

8.2.2 Dotted charts

The dotted chart plots every event as a dot at its time and its case [40]. With relative times (time since the start of the case) it shows how long the cases take and when the events happen within them; with absolute times it shows bursts, shifts and seasonality. A highlighted pattern is drawn as a line through its earliest occurrence in each case, which shows its typical position and duration. The duration of the occurrences of several patterns is summarized in a box plot per pattern; with `gap_unit="time"` the same durations become span constraints for mining (section 2.2.1).

8.2.3 Sequence logos

Aligning the sequences at the first occurrence of an anchor α gives, at every relative position j (negative before α, positive after), a distribution $p_j(e)$ of the events found there. The sequence logo stacks the events of each column with heights $p_j(e)\, R_j$, where

$$R_j = \log_2 K - H_j, \qquad H_j = -\sum_e p_j(e) \log_2 p_j(e), \tag{8.1}$$

is the information content of the column in bits and K the size of the alphabet [41]. Conserved positions — steps that almost always come at that distance from the anchor — are tall, variable positions are low. The logo is the aligned counterpart of the continuation-entropy tree of section 5.2.3: the tree measures uncertainty

given the whole prefix, the logo given only the distance from the anchor. Without an anchor the columns are the first positions of the sequences.

8.2.4 Arc diagrams

The arc diagram of Wattenberg draws one long sequence on a line and joins the consecutive non-overlapping occurrences of every maximal repeated contiguous subsequence by a semicircle, whose thickness is the length of the repeat [42]. Large arcs between distant parts reveal the form of a piece of music, a text or a long log: a refrain returns, a bridge stands apart. It complements the frequent sequences of a database: it finds what repeats *within* one sequence.

8.2.5 Episodes

Many alarm and event logs have no natural case identifier: they are one long stream of timestamped events. Mannila, Toivonen and Verkamo defined *episodes* for this setting [38]. A *serial* episode is an ordered sequence of events, a *parallel* episode a set of events in any order. Given a window width w, WINEPI slides a window $[s, s + w)$ over the stream and defines the frequency of an episode as the share of the windows that contain it; MINEPI counts its *minimal occurrences*, time intervals $[t_s, t_e]$ of width below w that contain the episode and no shorter interval inside them does. A window $[s, s + w)$ contains the occurrence $[t_s, t_e]$ exactly when $t_e - w < s \leq t_s$, so the WINEPI frequency of a serial episode is the total length of the union of these intervals over its minimal occurrences, divided by the length $T + w$ of the range of window starts; `spmviz` computes it this way, without materializing the windows. Serial episodes are grown one event at a time from the minimal occurrences of their prefixes, parallel episodes level by level, as in GSP. The *episode timeline* draws the minimal occurrences of the top episodes as segments on the time axis, one row per episode.

8.2.6 Summaries by minimum description length

The minimum description length (MDL) principle selects the set of patterns that compresses the data best: the best model is the one that minimizes the length of the model plus the length of the data encoded with it. Krimp applied the principle to itemsets [68], SQS [69] and SQUISH [70] to event sequences, and Sequence Synopsis to the visual summary of event data [71]. `spmviz` implements a transparent variant. A *code table* CT contains the single events and a set of contiguous patterns. Every sequence is covered greedily from left to right by the code table, longest patterns first; the *usage* $u(X)$ of a code X is the number

Algorithm 8.1 Greedy MDL summary

Input: database $\mathcal{D}$, candidate contiguous patterns C, maximum number of patterns k
Output: code table $\mathcal{CT}$ and the covers of the sequences

$\mathcal{CT} \leftarrow$ singletons; $L^* \leftarrow L(\mathcal{CT}, \mathcal{D})$;
foreach $X \in C$ *in order of decreasing* $\sigma(X)\,(|X|-1)$ **do**
  cover $\mathcal{D}$ with $\mathcal{CT} \cup \{X\}$ (left to right, longest code first) and compute L by eq. (8.2);
  if $L < L^*$ **then** $\mathcal{CT} \leftarrow \mathcal{CT} \cup \{X\}$, $L^* \leftarrow L$;
  if $\mathcal{CT}$ *has* k *patterns* **then break**;
foreach *pattern* $X \in \mathcal{CT}$ **do**
  if *removing* X *shortens* L **then** remove X;

In a nutshell Sequence views and summaries

- Index plots show every case; sorted and annotated by patterns they show where a pattern occurs.
- Chronograms show the distribution of events by position, frequency plots the most common complete sequences.
- Dotted charts add time; sequence logos align the cases at an anchor and measure how conserved each step is.
- Arc diagrams find the repeats within one long sequence.
- Episodes are the patterns of a stream without cases; MDL summaries replace hundreds of patterns by a few that compress the data.

of times it is used in the covers, and its code length is the Shannon code length $-\log_2\big(u(X)/\sum_Y u(Y)\big)$. The total description length is

$$L(\mathcal{CT}, \mathcal{D}) = L(\mathcal{CT}) + L(\mathcal{D} \mid \mathcal{CT}), \qquad L(\mathcal{D} \mid \mathcal{CT}) = \sum_X u(X)\Big(-\log_2 \frac{u(X)}{\sum_Y u(Y)}\Big), \tag{8.2}$$

where $L(\mathcal{CT})$ encodes every pattern of the table by the codes of its events in the singleton table and by its own code. Candidate patterns, the contiguous frequent sequences, are tried in order of decreasing $\sigma(X)\,(|X|-1)$, the number of codes a pattern could save in the covers, and kept when they shorten L; a final pass removes patterns whose removal shortens it further. The result is a handful of patterns that together explain the data, with the compression ratio $L(\mathcal{CT}, \mathcal{D})/L(\mathcal{CT}_0, \mathcal{D})$ against the singleton-only table $\mathcal{CT}_0$ as a measure of how much structure they capture. The *cover plot* shows the sequences as rows and colors every run of elements by the pattern that covers it.

8.3 Using it in spmviz

`plot_sequence_index(db, sort, highlight)`, `plot_state_distribution`, `plot_sequence_frequency`, `plot_dotted_chart(db, highlight)`, `plot_gap_distribution`, `plot_sequence_logo(db, anchor)` and `plot_arc_diagram(sequence)` draw the sequence views. `mine_episodes(events, times, window, ...)` with `min_frequency` (WINEPI) or `min_count` (MINEPI) and `kind` returns an `EpisodeSet` (a `PatternSet` with frequencies and minimal occurrences), drawn by `plot_episode_timeline`. `mdl_summary(db, min_support, max_length, max_patterns)` returns an `MDLSummary` with the selected patterns, their usages, the covers and the description lengths, drawn by `plot_cover`.

8.4 Examples

Index plots. Figure 8.1 shows eighty synthetic alarm floods sorted by two patterns, the pump cascade ⟨P-LOWFLOW P-TRIP OP-PUMPSTART⟩ and the cooling pair ⟨C-HITEMP C-VALVE⟩; their earliest occurrences are marked, and the floods that contain them come first. Sorted by length (fig. 8.2), the same plot shows how long the floods are.

```
plot_sequence_index(db, max_sequences=80, sort="pattern",
                    highlight=["P-LOWFLOW P-TRIP OP-PUMPSTART", "C-HITEMP
    C-VALVE"], max_length=14)
```

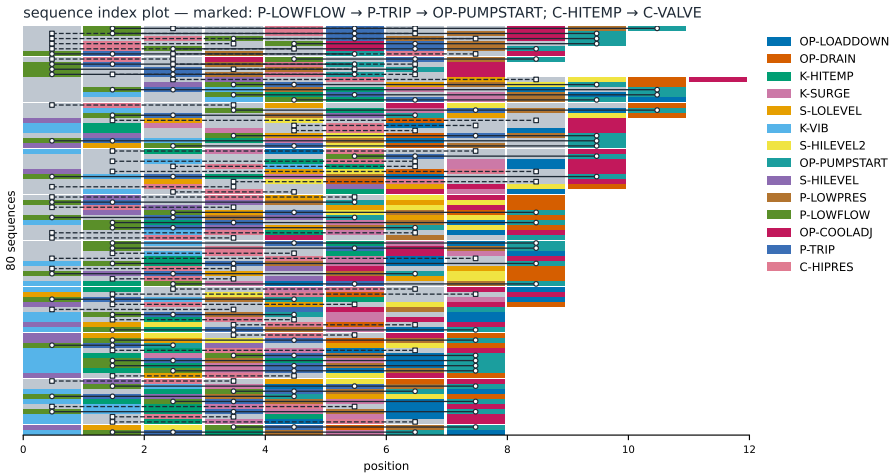


Figure 8.1. Index plot of eighty alarm floods sorted by two highlighted patterns, whose earliest occurrences are marked.

Chronogram and frequency plot. The chronogram of the daily routines (fig. 8.3) shows the day unfolding position by position: waking up and coffee at the start,

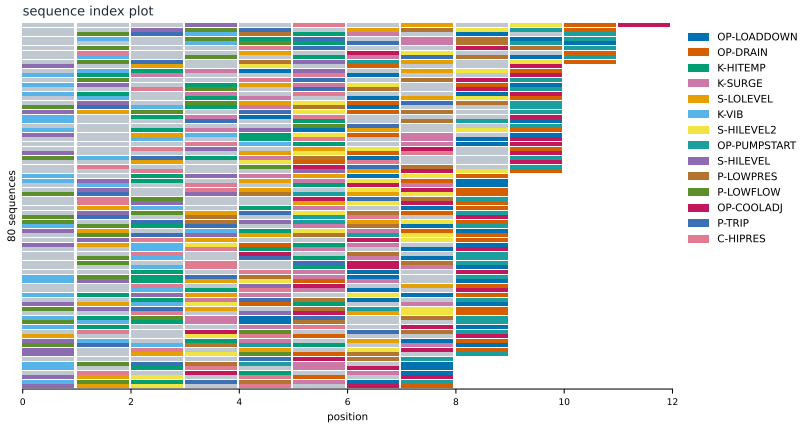


Figure 8.2. The same floods sorted by length.

work in the middle, dinner, television and sleep at the end; the entropy line is low at the ends of the day and high in its middle. The sequence frequency plot of the web shop (fig. 8.4) shows that the ten most frequent complete journeys make up 59% of all journeys.

```
plot_state_distribution(days, max_steps=15)
plot_sequence_frequency(shop, top=10)
```

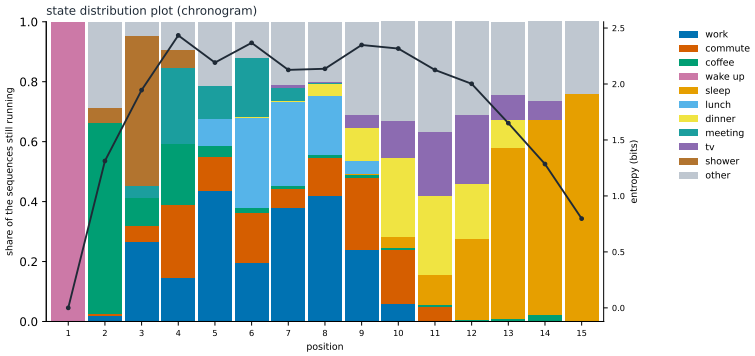


Figure 8.3. The state distribution plot (chronogram) of the daily routines, with the cross-sectional entropy (line, right axis).

Dotted chart. Figure 8.5 plots the events of 120 synthetic alarm floods against the time since the start of the flood; the line through the pump cascade shows where in the flood it happens and how long it takes. Figure 8.6 summarizes the time spans of the occurrences of the most frequent patterns as box plots.

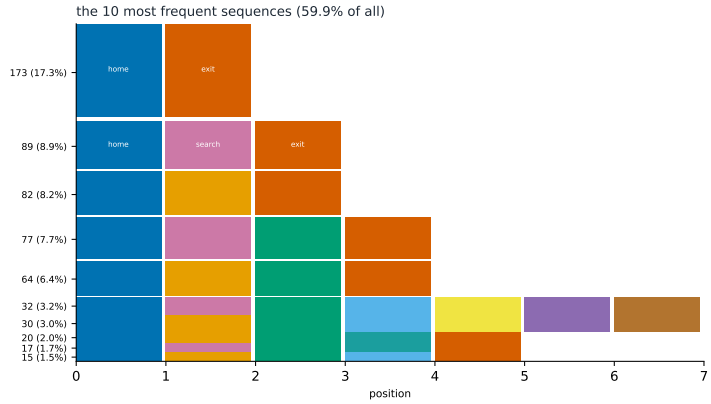


Figure 8.4. The ten most frequent complete web-shop journeys, bar heights proportional to their frequency.

```
plot_dotted_chart(db, max_sequences=120, highlight="P-LOWFLOW P-TRIP
    OP-PUMPSTART")
```

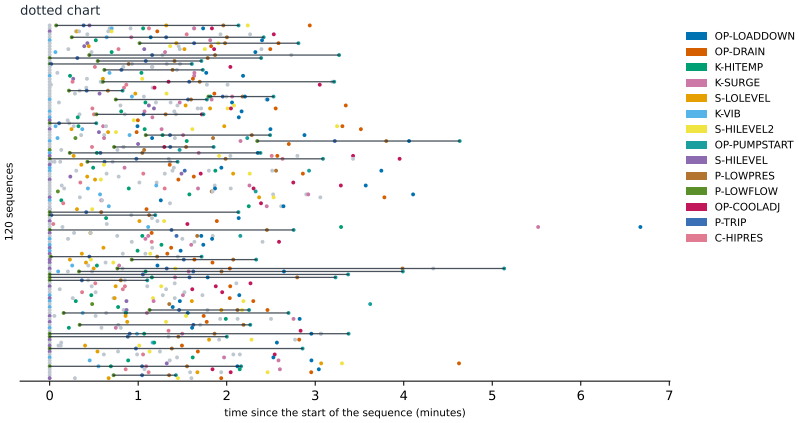


Figure 8.5. Dotted chart of 120 alarm floods (time since the start of each flood); the line follows the pump cascade.

Sequence logos. Around the first visit to the cart (fig. 8.7) the logo is tallest at the cart itself and at the position just before it, mostly a product page; the positions further away are more variable. The logo of the first notes of the melody phrases (fig. 8.8) shows the probabilities of the notes, without information weighting.

```
plot_sequence_logo(shop, anchor="cart", before=3, after=4)
```

Arc diagrams of melodies. The arc diagram of Beethoven's *Ode to Joy* (fig. 8.9) shows its form at once: the opening phrase returns, joined by the largest arcs,

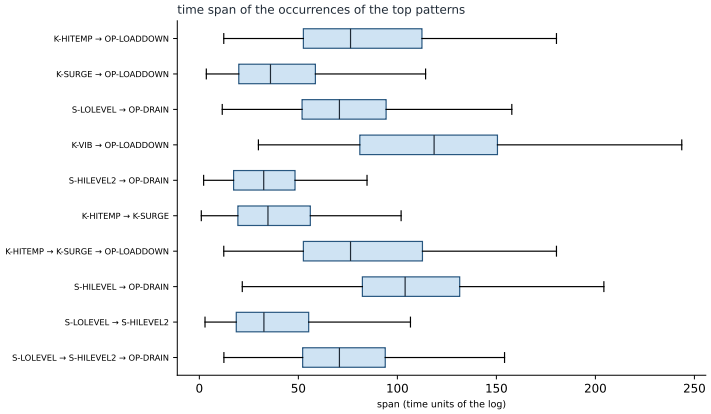


Figure 8.6. Time spans of the occurrences of the most frequent patterns.

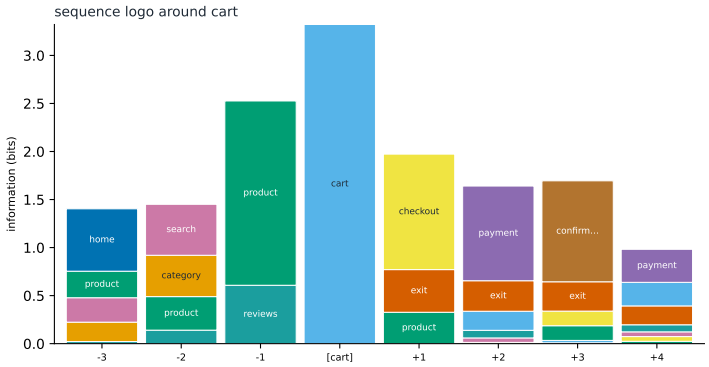


Figure 8.7. Sequence logo of the web-shop journeys around the first visit to the cart; heights in bits.

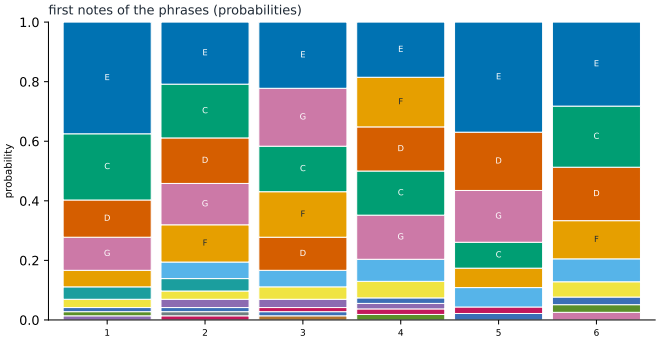


Figure 8.8. The first notes of the phrases of thirteen songs (probabilities).

while the bridge stands apart. *Twinkle, Twinkle, Little Star* (fig. 8.10) repeats its first phrase at the end and its middle phrase twice.

```
ode = songs[titles.index("Ode to Joy")]
plot_arc_diagram([e[0] for e in ode], min_length=3, title="Ode to Joy:
    repeats of at least three notes")
```

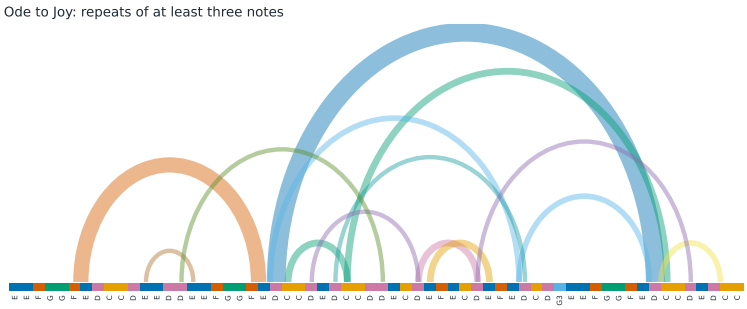


Figure 8.9. Arc diagram of *Ode to Joy*: repeats of at least three notes.

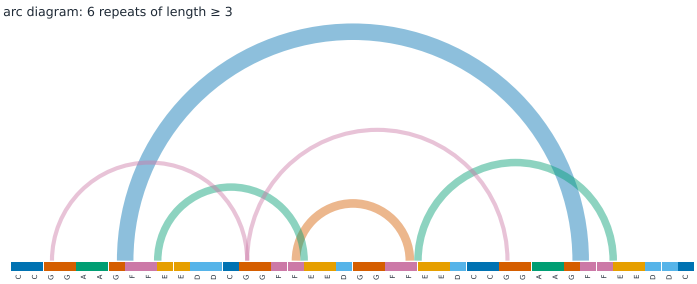


Figure 8.10. Arc diagram of *Twinkle, Twinkle, Little Star*.

Episodes of an alarm stream. The busiest day of the alkylation plant, 1,150 events, is treated as one stream with times in minutes. With a ten-minute window, the most frequent serial episode of two events is ⟨O1132 O1136⟩, contained in 8% of the windows, with 12 minimal occurrences; fig. 8.11 draws the minimal occurrences of the eight most frequent serial episodes.

```
ep = sv.mine_episodes(events, times=minutes, window=10,
    min_frequency=0.02, max_length=3)
multi = ep.filter(min_length=2)
print(multi.to_frame().sort_values("frequency",
    ascending=False).head(8)[["pattern", "frequency",
    "minimal_occurrences"]])
plot_episode_timeline(ep, top=8)
```

A summary in a few patterns. From the contiguous frequent sequences of the daily routines, the MDL summary keeps eight patterns that shorten the description

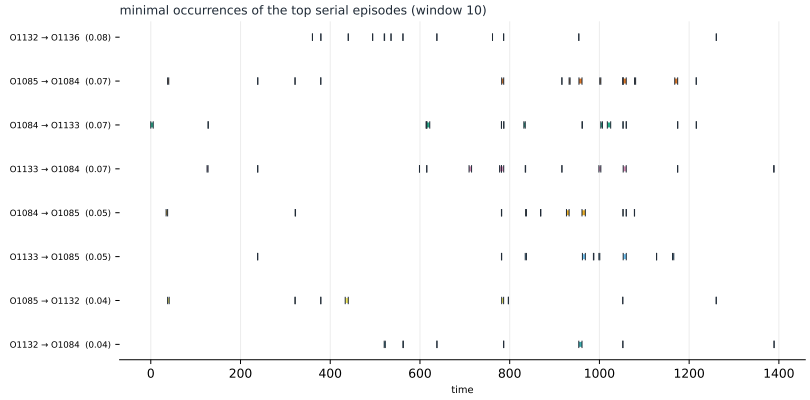


Figure 8.11. Minimal occurrences of the eight most frequent serial episodes of the busiest day of the alarm log (window ten minutes); the numbers are WINEPI frequencies.

to 76% of the singleton code (fig. 8.12); among them are the three persona-specific mornings and the evenings ⟨work commute gym dinner tv sleep⟩. Allowing 2, 5, 10 or 20 patterns gives compression ratios of 0.93, 0.90, 0.77 and 0.73: the first patterns explain most of the structure.

```
summary = sv.mdl_summary(db, min_support=0.1, max_length=8,
    max_patterns=12)
print(f"{len(summary.patterns)} patterns, description length
    {summary.compression:.0%} of the singleton code")
print(summary.to_frame())
plot_cover(db, summary, max_sequences=40, max_length=15)
```

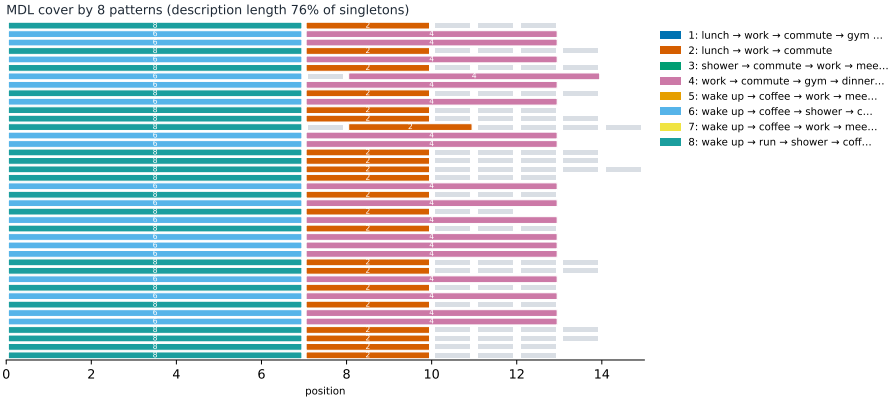


Figure 8.12. The MDL cover of forty days: every run of activities is colored by the summary pattern that covers it.

CHAPTER

9



Interoperability

> An analysis rarely starts and ends in one package. This chapter shows how event data enter `spmviz` — from pandas tables, SPMF files, XES logs and pm4py — how other miners can be used in place of the built-in ones, and how patterns, measures and graphs leave the package for pandas, spreadsheets, SPMF, NetworkX, Gephi, Cytoscape, igraph and the web.

9.1 Problem and idea

Sequential pattern mining has excellent tools: the SPMF library implements more than two hundred algorithms in Java [28], pm4py is the process-mining library of Python [80], Seq2Pat solves constraint-based mining with decision diagrams [26], and network analysis has NetworkX, igraph, Gephi and Cytoscape [23, 78, 79]. The original NBVFS scripts already combined SPMF, pandas, NetworkX and pyvis through intermediate CSV files [5]. `spmviz` replaces the files by two objects with a stable interface, `SequenceDatabase` and `PatternSet`, and by NetworkX graphs whose nodes carry every attribute a view uses. Whatever reads or writes these three kinds of objects can be combined with every view of the book (fig. 9.1).

9.2 Data in

Most event data arrive as a long table with a case identifier, an activity and a timestamp. `SequenceDatabase.from_event_log(df, case, activity, timestamp)` groups the rows by case, orders them by time (or keeps the order of the table with `sort=False`), merges events with equal timestamps into itemsets if asked, keeps the timestamps in seconds and the other columns of the first row of every case as attributes. Timestamps may be strings, Python or pandas

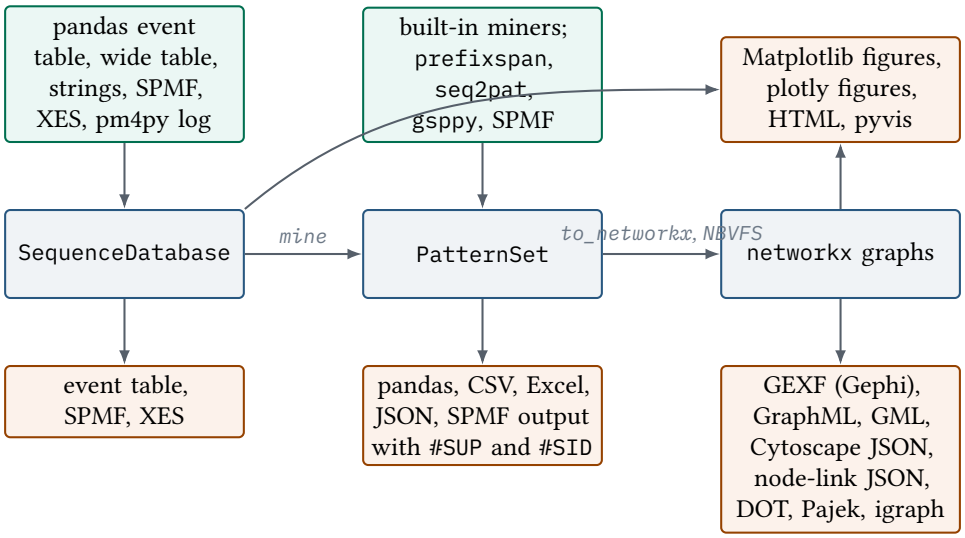


Figure 9.1. How data enter and leave `spmviz`. Three kinds of objects carry an analysis: sequence databases, pattern sets and NetworkX graphs; every reader produces one of them and every view consumes them.

datetimes, time-zone aware or not; aware ones are converted to UTC. `from_wide` reads one sequence per row, `from_strings` one per string, `from_spmf` the SPMF input format (with the `@CONVERTED_FROM_TEXT` header for text items), `io.read_xes` the IEEE XES standard of process mining, and `from_pm4py` a pm4py event log or data frame. `to_event_log`, `io.write_spmf` and `io.write_xes` write a database back.

```
log = pd.DataFrame({
    "case":     [1, 1, 1, 2, 2, 3, 3, 3],
    "activity": ["login", "search", "buy", "login", "search", "login",
    "search", "buy"],
    "time":     pd.to_datetime(["2026-01-01 10:00", "2026-01-01 10:02",
    "2026-01-01 10:09",
                                "2026-01-02 09:00", "2026-01-02 09:01",
                                "2026-01-03 18:00", "2026-01-03 18:05",
    "2026-01-03 18:06"]),
})
db = sv.SequenceDatabase.from_event_log(log, case="case",
    activity="activity", timestamp="time")
ps = sv.mine(db, min_support=2)
print(ps.to_frame()[["pattern", "support", "confidence"]])
```

9.3 Other miners

`sv.mine(db, ..., algorithm=name)` also runs external miners and returns their result as a `PatternSet`; the supporting sequences are computed when the miner does not report them, so every view works on the result. Four back ends are available (`spmviz.backends.available()` lists the installed ones):

- `"prefixspan-py"`: the `prefixspan` package [25], with closed, generator and top-k variants; sequences of single items, no gap constraints;
- `"seq2pat"`: Seq2Pat [26], constraint-based mining of sequences of single items;
- `"gsppy"`: the GSP-Py package [27]; version 5.3 returned the same supports as the built-in miners but missed some patterns that repeat an item, such as ⟨product product⟩;
- `"spmf"`: the SPMF library [28], called through Java with a user-supplied `spmf.jar` (CM-SPAM, PrefixSpan, or any algorithm with explicit arguments); `backends.spmf_command` prints the command line, so that SPMF can also be run by hand and its output read with `io.read_patterns`.

Comparing a back end with the built-in miners is a one-line check (section 2.6): two pattern sets are identical when their patterns and supports are.

9.4 Results out

`ps.to_frame()` returns the patterns as a pandas data frame with every measure; `ps.write(path)` writes CSV, JSON, Excel or the SPMF output format with `#SUP` and `#SID` fields, chosen by the extension, and `io.read_patterns` reads them back, with the supporting sequences when present. Text items are coded in SPMF files with `@ITEM` lines, so items with spaces survive the round trip.

```
sv.io.write_spmf(db, tmp / "db.txt")                # SPMF input format
    (text items coded)
ps.write(tmp / "patterns.txt")                      # SPMF output: #SUP
    and #SID
ps.write(tmp / "patterns.json")
back = sv.io.read_patterns(tmp / "patterns.json", db)
print((tmp / "patterns.txt").read_text())
```

Every network of the book is a NetworkX graph: the extension network, the lattice and the similarity graph of a pattern set (`ps.to_networkx(kind)`), the fitted `NBVFS` views, prefix trees, transition graphs and higher-order networks. Their nodes carry the support, relative support, length, confidence, cumulative confidence, start event and added event of the patterns, and the layouts store coordinates `x` and `y`. `io.write_graph` (and `NBVFS.write`) chooses the format by the extension (table 9.1); GEXF files carry Gephi's position, color and size attributes, so a map opens in Gephi as it was drawn. `io.to_igraph` and `io.from_igraph` convert to and from igraph.

```
G = ps.to_networkx("extension")                    # a networkx.DiGraph
print(G.nodes["login → search"])
wn = sv.NBVFS("wn").fit(ps)
for ext in ("gexf", "graphml", "cyjs", "json"):
    wn.write(tmp / f"network.{ext}")               # with x, y, support,
    confidence, ...
H = sv.io.read_graph(tmp / "network.graphml")
print(sorted(H.nodes(data=True))[0])
```

Table 9.1. Graph formats written by `io.write_graph` and `NBVFS.write`.

Extension	Format	Typical use
`.gexf`	GEXF 1.2 with `viz` attributes	Gephi, with positions, colors and sizes
`.graphml`	GraphML	yEd, Cytoscape, igraph, R
`.gml`	GML	older network tools
`.cyjs`	Cytoscape JSON with positions	Cytoscape and Cytoscape.js
`.json`	NetworkX node-link JSON	web pages, D3
`.dot`	Graphviz	typeset drawings
`.net`	Pajek	Pajek, older tools
`.csv`, `.xlsx`	node and edge tables	spreadsheets, databases

9.5 Interactive figures

The static views of the book are Matplotlib figures, which suit print and reports. For exploration, `spmviz.interactive` returns plotly figures: zoomable sunbursts, icicles and treemaps of the prefix tree, Sankey diagrams, and NBVFS maps and other networks with hover information on every node. `fig.write_html(path)` writes a standalone page. `interactive.to_pyvis` reproduces the draggable physics view of the original NBVFS scripts [5]. In the online documentation every example runs as a Jupyter notebook, so the interactive figures can be explored there as well.

```
fig = interactive.sunburst_figure(db, max_depth=6)
fig
```

In a nutshell Interoperability

- Event tables, wide tables, strings, SPMF, XES and pm4py logs become a `SequenceDatabase`.
- The `prefixspan`, `seq2pat` and `gsppy` packages and SPMF can replace the built-in miners; their results are `PatternSets`.
- Patterns go to pandas, CSV, Excel, JSON and SPMF; graphs to GEXF, GraphML, Cytoscape JSON and six other formats, with all attributes and positions.
- Interactive plotly and pyvis versions exist for trees, flows and maps.

CHAPTER

10



Benchmarks and case studies

> This chapter puts the methods to work. It first measures the built-in miners on the public benchmark databases of the sequential pattern mining literature, then follows two analyses from the raw log to an interpretation: the click-stream of an online benefit application and the alarm log of a chemical plant, the two logs of the NBVFS paper. It closes with a compact procedure for the analysis of a new log.

10.1 Problem and idea

A toolbox is useful if it is fast enough for real databases and if its views lead to conclusions that can be checked. The first question is answered by benchmarks: the databases distributed with SPMF [28] are the common yardstick of the field, and running the five miners of the package on them, together with an independent implementation, tests correctness and speed at once. The second question is answered by case studies in which several views of the same data are combined, each answering the question it answers best (table 3.1), and in which the conclusions are confronted with the data.

10.2 Benchmark databases

`datasets.fetch_spmf(name)` downloads and caches nine databases distributed with SPMF (table 10.1). They cover the main application areas of sequential pattern mining: click streams with short sequences over medium alphabets (BMS1, BMS2, KOSARAK, MSNBC) or long sequences (FIFA), text with very large alphabets (BIBLE, LEVIATHAN), sign language (SIGN) and spatial trips (BIKE). The package first tries the SPMF web site and then public mirrors of the same files; for this book the files were obtained from the mirrors. Utility annotations of mirror files are

dropped. The statistics of table 10.1 were computed from the files as read; they agree with those published by SPMF except for BMS1 (average length 2.5 against 2.4), BIKE (69 items against 67) and FIFA (average length 36.2 against 34.7), whose mirror is the sequential-utility version of the database. Pattern counts on these three databases are therefore not directly comparable with counts published for the SPMF files.

Table 10.1. Benchmark databases available through `fetch_spmf`: number of sequences N, number of items $|\mathcal{I}|$ and average length, computed from the files used in this book (— for the two databases that were not benchmarked).

Name	N	$\lvert\mathcal{I}\rvert$	length	Content
BIBLE	36,369	13,905	21.6	the Bible, one sentence per sequence, words as items
BMS1	59,601	497	2.5	BMS-WebView-1 (Gazelle) click streams of the KDD Cup 2000 [81]
BMS2	—	—	—	BMS-WebView-2 (Gazelle) click streams of the KDD Cup 2000 [81]
FIFA	20,450	2,990	36.2	click streams of the FIFA World Cup 1998 web site
KOSARAK10K	10,000	10,094	8.1	the first 10,000 click streams of a Hungarian news portal (FIMI repository)
LEVIATHAN	5,834	9,025	33.8	T. Hobbes, *Leviathan* (1651), one sentence per sequence
MSNBC	—	—	—	page-category click streams of msnbc.com [82], short sequences removed
SIGN	730	267	52.0	American Sign Language utterances [83]
BIKE	21,078	69	7.3	stations of shared-bike trips

10.3 Run times

Table 10.2 reports the run times of the four exact miners that need no candidate generation and of the `prefixspan` package [25] as an external reference, at thresholds that give a few hundred to a few thousand patterns; table 10.3 repeats the measurement with the gap constraint $g_{\max} = 1$ and adds GSP. The measurements were taken with `tools/benchmark.py` under Python 3.13 on one core of an Intel Xeon processor at 2.1 GHz; they are single runs, so differences below about ten percent are not meaningful.

Three observations hold across the databases. First, all algorithms return identical pattern sets, the external package included. Second, without constraints the bitmap algorithms are the fastest: SPAM in five of the seven cases and CM-SPAM in the other two, between about two and nine times faster than PrefixSpan and SPADE. A Python integer packs the bitmaps of all sequences, so one shift and one intersection process the whole database; PrefixSpan and SPADE loop over

Table 10.2. Run times (seconds) of the built-in miners and of the `prefixspan` package on the benchmark databases, maximum length 6, no gap constraint. All algorithms return identical pattern sets; the fastest time of each row is in bold; — marks databases with itemsets or more than 300,000 elements, on which the pure-Python `prefixspan` package was not run.

Database	$\sigma_{\min}$	patterns	PrefixSpan	CM-SPAM	SPAM	SPADE	`prefixspan`
SIGN	40%	518	0.39	0.20	**0.10**	0.79	5.99
LEVIATHAN	10%	651	2.57	0.80	**0.58**	2.05	49.14
BIKE	1%	901	1.09	0.73	**0.63**	2.52	—
BMS1	0.2%	798	1.70	**0.94**	0.95	4.16	175
KOSARAK10K	1%	392	0.57	0.23	**0.18**	0.50	23.15
FIFA	15%	5,110	85.22	**15.33**	15.68	141	—
BIBLE	10%	174	5.08	1.41	**0.78**	5.62	—

Table 10.3. Run times (seconds) with the gap constraint $g_{\max} = 1$ (contiguous patterns) at half the threshold of table 10.2; the three algorithms return the same number of patterns. GSP was not run on BIBLE, BMS1 and KOSARAK10K.

Database	$\sigma_{\min}$	patterns	PrefixSpan	CM-SPAM	GSP
SIGN	20%	98	0.07	0.06	22.36
LEVIATHAN	5%	106	0.28	0.25	13.70
BIKE	0.5%	365	0.38	0.34	10.55
BMS1	0.1%	672	0.38	0.62	—
KOSARAK10K	0.5%	228	0.13	0.14	—
FIFA	7.5%	151	1.33	0.99	310
BIBLE	5%	65	1.27	0.78	—

sequences and occurrences in the interpreter. The `prefixspan` package, also pure Python, is 60 to 185 times slower than SPAM on the four databases on which it ran. Third, with a gap constraint the search space is much smaller: PrefixSpan and CM-SPAM finish in at most 1.3 seconds, and the level-wise GSP is 30 to 370 times slower. Pure-Python miners cannot compete with the compiled implementations of SPMF on very large databases, but at the thresholds that produce readable pattern sets — a few hundred patterns — they run in seconds, which is what interactive visual analysis needs.

```
try:
    db = sv.datasets.fetch_spmf("SIGN")
except OSError as exc:  # no network: fall back to a synthetic database of
    the same size
    print("download failed:", exc)
    db = sv.datasets.make_quest(730, n_items=267, avg_elements=52,
    avg_itemset=1.0, random_state=0)
print(db.stats())
ps = sv.mine(db, min_support=0.5, max_length=5)
print(ps)
```

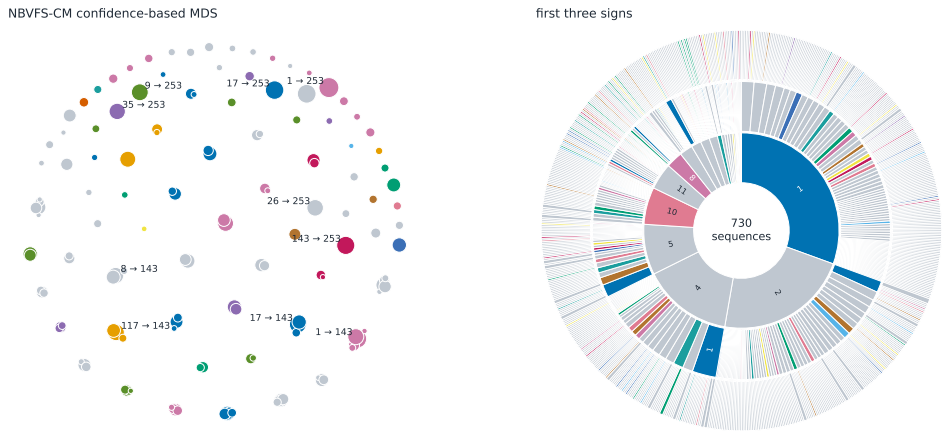


Figure 10.1. The SIGN benchmark: the CM map of its frequent sequences of length at least two (left) and the sunburst of the first three signs of its 730 utterances (right).

10.4 Case study: an online benefit application

The click-stream of the NBVFS paper records 343 sessions of applicants for unemployment benefit on the portal of the Dutch Employee Insurance Agency, part of the BPI Challenge 2016 data [29]. Every click opens one of 14 services of the application, among them Your personal details, Your last employer, The dismissal, Hours worked, Your situation, Your income and The labor market. Where do the applicants start, which paths are typical, where do they hesitate, and which next step can be predicted?

The Sankey diagram of the first five steps (fig. 10.2) shows that half of the sessions start with Your personal details (90 sessions) or Your last employer (84), and that the paths diverge quickly. The directly-follows graph (fig. 10.3) shows the services that applicants revisit, and the continuation-entropy tree (fig. 10.4) where the next service is hard to predict. The 241 frequent sequences mined as in the paper (section 4.4) were drawn as WN, CM and TM views in chapter 4; their measure scatter plot (fig. 7.7) singled out the repeated visits of Your situation. Finally, the prediction view (fig. 10.5) answers the operational question: after ⟨Your last employer The dismissal⟩, the longest frequent sequence that matches the session so far, the most likely next services are The dismissal again (confidence 0.45, 26 sessions) and Hours worked (0.34, 20 sessions). Repeated visits of the same service are frequent in this log: 55% of the sessions open the same service twice in a row. Whether this reflects multi-page forms or applicants going back to correct their entries cannot be told from the log alone, but it is the first question to ask the owners of the application.

```
obs = [0, 1]  # Your last employer, The dismissal
print(sv.predict_next(ps, obs))
plot_prediction(ps, obs)
```

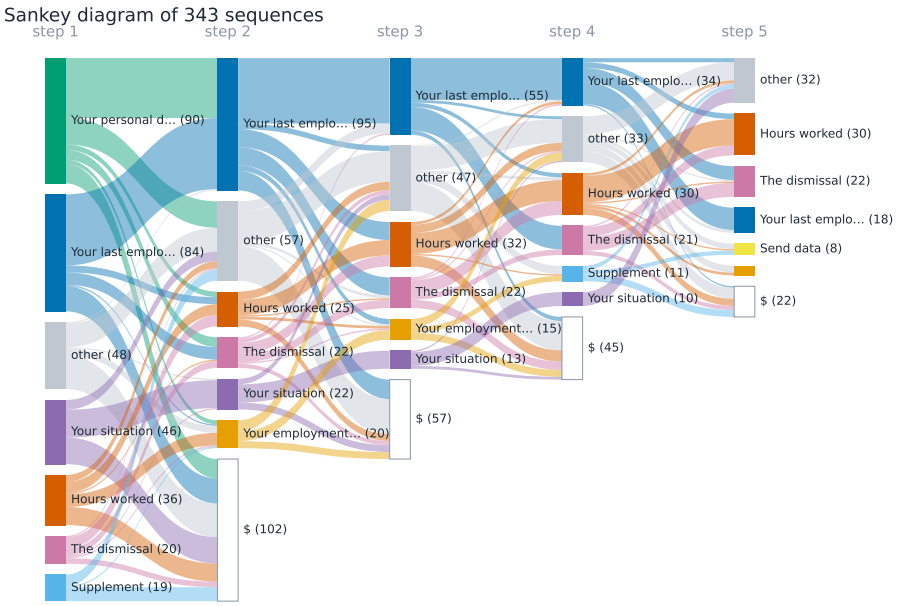


Figure 10.2. The first five steps of the 343 sessions as a Sankey diagram.

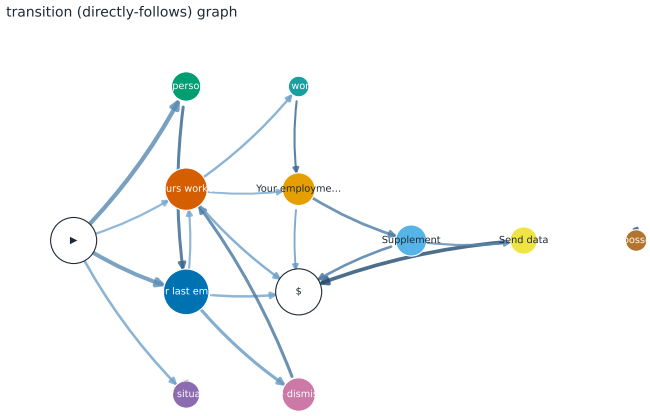


Figure 10.3. The directly-follows graph of the 14 services.

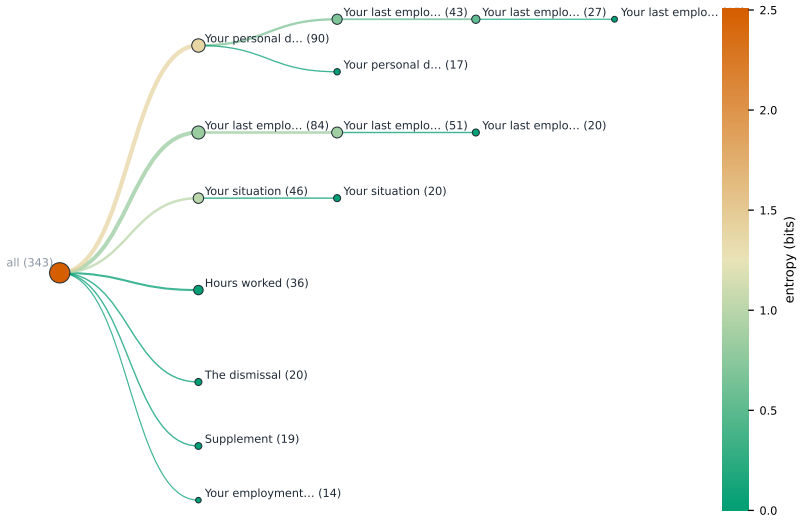


Figure 10.4. The continuation-entropy tree of the first four steps.

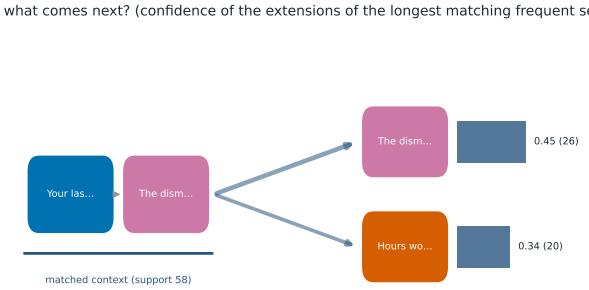


Figure 10.5. What comes next after ⟨Your last employer The dismissal⟩: the extensions of the longest matching frequent sequence, with their confidences and supports.

10.5 Case study: early warning in alarm management

The alarm log of the alkylation plant (section 4.4) has 3,956 traces of alarms and operator actions. The research of the group used frequent sequences of such logs to suppress alarms that follow others predictably [2], to predict the next alarm by sequence compression and alignment [3], and to design operator interfaces [4]. Two of the views of this book support an operator directly.

The scenario tree of the most frequent start event, O1084 (fig. 10.6), is the pattern tree of the frequent sequences that start with it, reduced to the three most frequent continuations of every node; it lists the typical scenarios after this action, with their supports. The prediction view (fig. 10.7) works online: given the events

seen so far, it finds the longest frequent sequence that matches their end and ranks its extensions by confidence. After ⟨O1084 O1085⟩, the start of the most supported frequent sequence of length at least three, the action O1084 follows in 52% of the cases, O1085 in 25%, O1133 in 24%. The confidences do not add up to one: with a maximum gap of two, a trace may continue the sequence in several ways. A first check of the predictions on the 421 traces that contain the observed pair gives a top-1 accuracy of 0.38. The check is in-sample, because the patterns were mined from the same traces, so it is optimistic; an honest evaluation would mine on one part of the log and predict on another, as in the alignment-based study of the alarm log [3].

```
long = ps.filter(min_length=3).sort("support").patterns[0]
observed = [e[0] for e in long[:2]]
print("observed:", [db.label(i) for i in observed])
print(sv.predict_next(ps, observed))
plot_prediction(ps, observed)
```

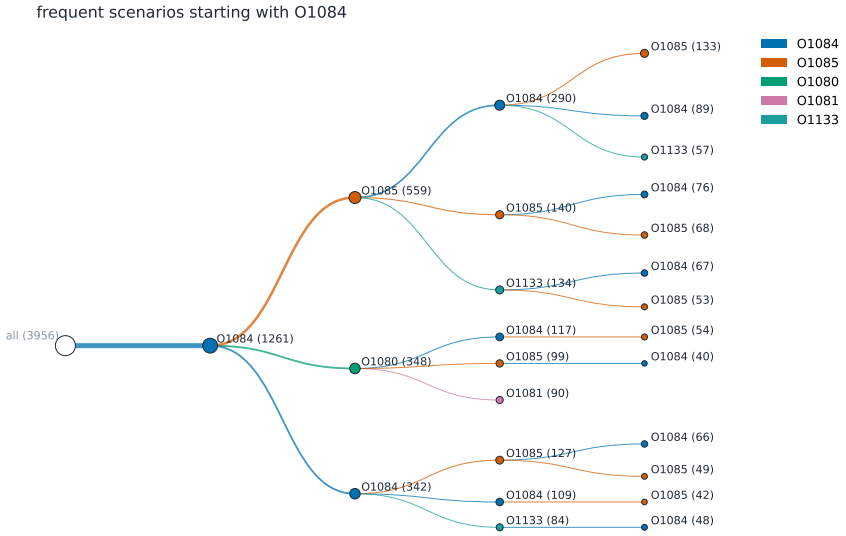


Figure 10.6. Frequent scenarios starting with the operator action O1084: the three most frequent continuations of every node, edge width proportional to the support.

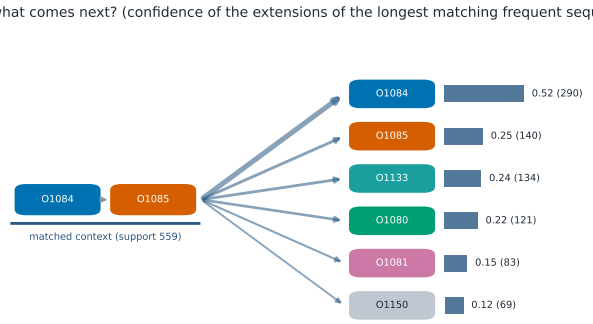


Figure 10.7. Online prediction after ⟨O1084 O1085⟩: the extensions of the longest matching frequent sequence ranked by confidence.

10.6 Analysing a new log

The examples of the book suggest a procedure for a new log (fig. 10.8). Its steps can be repeated in any order, and every step has a view that checks the previous one.

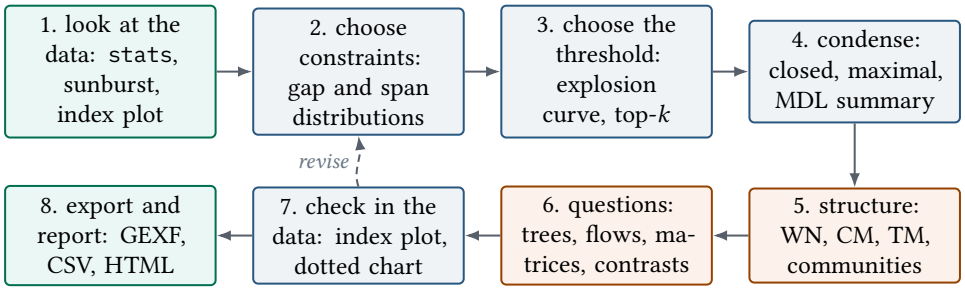


Figure 10.8. A procedure for the analysis of a new log. The dashed arrow is the most frequent iteration: a pattern seen in the data that does not match its picture usually means that a constraint was too loose or too tight.

1. *Look at the data before mining.* The statistics of the database, a sunburst of the beginnings and an index plot of a sample show the lengths, the alphabet and the obvious regularities.
2. *Choose the constraints.* The distributions of the gaps and spans between events (`plot_gap_distribution`) tell which distances are meaningful; most logs need a gap constraint, and logs with timestamps often a time window.
3. *Choose the threshold.* The explosion curve (`measures.pattern_counts`) shows how the number of patterns changes with the threshold; a few hundred patterns are readable, a few thousand need condensation. Top-k mining avoids the choice.

In a nutshell Benchmarks and case studies

- On the SPMF benchmarks the five miners and the `prefixspan` package return identical pattern sets; SPAM and CM-SPAM are the fastest in pure Python.
- At thresholds that produce readable pattern sets, all databases are mined in seconds.
- In the click-stream, the views locate the typical starts, the revisited services and the hard-to-predict steps; in the alarm log, scenario trees and online prediction support the operators.
- Conclusions drawn from a picture should be checked in the sequences themselves.

4. *Condense.* Closed patterns lose nothing, maximal patterns lose the supports, an MDL summary keeps only the patterns that explain the data.
5. *Find the structure.* The WN shows how the sequences grow, the CM map which belong to the same scenario, the TM map and the communities which occur together.
6. *Answer the questions* with the views listed in table 3.1.
7. *Check in the data.* Every pattern that leads to a conclusion should be found in the sequences, with an index plot or a dotted chart.
8. *Export and report.* Graphs go to Gephi or Cytoscape, tables to spreadsheets, interactive figures to HTML pages.

APPENDIX

A



API reference

This appendix lists every public class and function of `spmviz`, module by module, with the first sentence of its docstring. The lists are generated from the `__all__` attribute of each module by `book/build_api_appendix.py`, so they are complete by construction; the full signatures, parameters, and attributes are in the online documentation and in the docstrings (`help(spmviz.NBVFS)`). The conventions of the API are described in sections 2.5 and 4.3, the notation of the docstrings in table 2.1.

A.1 **`spmviz.data`**: Sequence databases and patterns

Name	Kind	Description
`Constraints`	class	Gap and span constraints of an occurrence
`SequenceDatabase`	class	A database of sequences of itemsets with optional timestamps and attributes
`as_pattern`	function	Normalize a pattern given as tuple of tuples, list or string
`contains`	function	Whether a sequence contains a pattern under the given constraints
`find_occurrence`	function	Element indices of the earliest-ending occurrence of a pattern, or None
`format_pattern`	function	Format a pattern as text
`is_subpattern`	function	Whether pattern p is contained in pattern q (no gap constraints)
`item_key`	function	Sort key giving a total order over mixed items: numbers first, then strings
`parent_pattern`	function	The pattern with its last item removed (the empty tuple for single items)
`pattern_items`	function	Items of a pattern in order of appearance (with repetitions)
`pattern_length`	function	Number of items of a pattern

A.2 `spmviz.patterns`: Pattern sets

Name	Kind	Description
`PatternSet`	class	Frequent sequential patterns with their supports and supporting sequences

A.3 `spmviz.mining`: Mining algorithms

Name	Kind	Description
`ALGORITHMS`	constant	Names of the built-in mining algorithms
`EpisodeSet`	class	Frequent episodes with WINEPI frequencies and minimal occurrences
`closed_mask`	function	Boolean mask of the closed patterns of a pattern set
`cmspam`	function	Mine frequent sequential patterns with CM-SPAM (bitmaps + co-occurrence pruning)
`generator_mask`	function	Boolean mask of the generator patterns of a pattern set
`gsp`	function	Mine frequent sequential patterns with GSP (Apriori-style, any constraint)
`maximal_mask`	function	Boolean mask of the maximal patterns of a pattern set
`mine`	function	Mine frequent sequential patterns
`mine_episodes`	function	Frequent serial or parallel episodes of one event stream
`prefixspan`	function	Mine frequent sequential patterns with PrefixSpan
`sequential_rules`	function	Rules antecedent ⇒ consequent from every split of every pattern
`spade`	function	Mine frequent sequential patterns with SPADE (vertical id-lists)
`spam`	function	Mine frequent sequential patterns with SPAM (bitmaps, Apriori candidate pruning only)
`topk`	function	The k most frequent patterns (ties at the k-th support kept)

A.4 `spmviz.backends`: External back ends

Name	Kind	Description
`available`	function	Which external back ends can be used in this environment

Name	Kind	Description
`run`	function	Run an external miner and return a PatternSet (called by mine)
`spmf_command`	function	The SPMF command line for an algorithm

A.5 `spmviz.measures`: Measures, continuations, contrasts, similarities

Name	Kind	Description
`END`	constant	Token of the end of a sequence in continuation tables and trees
`content_similarity`	function	Similarity of the patterns themselves (their item strings)
`continuation`	function	Distribution of the element that follows an occurrence of a pattern
`continuation_entropy`	function	Entropy (bits) of the continuation distribution of a pattern
`continuation_table`	function	Continuation statistics of every pattern: entropy, normalized entropy, most likely next items
`contrast`	function	Support of every pattern in two or more groups of sequences
`entropy`	function	Shannon entropy (bits) of a probability vector or a pandas Series
`pattern_counts`	function	Number of all, closed and maximal patterns for a range of minimum supports
`pattern_measures`	function	Interestingness measures of every pattern
`time_statistics`	function	Span and gaps of the earliest occurrence of every pattern in every supporting sequence
`transaction_similarity`	function	Similarity of the supporting sequence sets of every pair of patterns

A.6 `spmviz.networks`: Graphs of sequences and patterns

Name	Kind	Description
`START`	constant	Token of the start of a sequence in transition graphs
`confidence_matrix`	function	Confidence adjacency **A**: a_{ij} is the weight of the extension $\varphi_i \to \varphi_j$
`double_tree`	function	Predecessor and successor trees of a pivot pattern (word tree / double tree)

Name	Kind	Description
`extension_network`	function	Weighted extension network of frequent sequences (NBVFS-WN)
`higher_order_network`	function	Variable-order network of the sequences (BuildHON; Xu, Wickramarathne and Chawla, 2016)
`item_graph`	function	Items as nodes, a → b weighted by the support of the 2-pattern ⟨a b⟩ (eventually-follows)
`pattern_communities`	function	Groups of patterns that tend to occur in the same sequences (sub-processes)
`pattern_lattice`	function	Hasse diagram of the containment order: p → q when q has one item more and contains p
`prefix_tree`	function	Trie of the sequences (the aggregation behind sunbursts, icicles, LifeFlow and word trees)
`similarity_graph`	function	Undirected graph of similar patterns (kNN and/or threshold on a similarity)
`similarity_matrix`	function	A pattern similarity by name: "transaction" (Jaccard), "confidence" (closure), "content" (LCS), or a precomputed square array
`sublogs`	function	The sequences supporting the patterns of each group: one sub-database per community
`transition_graph`	function	Directly-follows graph (first-order transitions; process mining DFG, Markov chain)
`transitive_confidence`	function	Max-product transitive closure $\mathbf{A}^*$ (NBVFS-CM)

A.7 `spmviz.embedding`: Embeddings, fusion and landscapes

Name	Kind	Description
`align`	function	Rotate/reflect and center Y to best match reference (orthogonal Procrustes)
`classical_mds`	function	Classical (Torgerson–Gower) scaling of a dissimilarity matrix
`fuse_similarities`	function	Combine pattern similarities: weighted "mean", "min", "max", "product" or "snf"
`landscape`	function	Weighted Gaussian kernel density of a 2-D map
`mds`	function	Embed a dissimilarity matrix; returns (Y, stress)
`smacof`	function	Metric MDS by majorization (SMACOF)
`snf`	function	Similarity network fusion of several similarity matrices (values in [0, 1])

A.8 spmviz.layout: Layouts

Name	Kind	Description
layered_layout	function	Layered (Sugiyama-style) layout with barycentric crossing reduction
radial_tree_layout	function	Concentric layout: the root in the center, depth d on the circle of radius d
tree_layout	function	Layered tree layout (tidy): leaves on a line, parents centered over their children

A.9 spmviz.nbvfs: NBVFS maps

Name	Kind	Description
NBVFS	class	Network-based visualisation of frequent sequences (WN, CM, TM and fused maps)

A.10 spmviz.summary: MDL summaries

Name	Kind	Description
MDLSummary	class	Result of mdl_summary
mdl_summary	function	Greedy MDL selection of contiguous patterns

A.11 spmviz.predict: Next-event prediction

Name	Kind	Description
match_context	function	Longest suffix of observed that is a pattern of ps with at least min_children extensions
predict_next	function	Most likely next events after an observed partial sequence

A.12 **spmviz.viz**: Static visualizations

Name	Kind	Description
BRAND	constant	Pannon blue of the figures (#285780)
CATEGORICAL	constant	Categorical palette of the item colors
OKABE_ITO	constant	The Okabe–Ito color-blind-safe palette
db_colors	function	Colors of the items of a database (most frequent first), keyed by label or item
embed_patterns	function	Coordinates of the patterns from a similarity (name, matrix, or a list of them to fuse)
item_colors	function	Stable item → color mapping
plot_arc_diagram	function	Arc diagram of the repeated subsequences of one long sequence (Wattenberg, 2002)
plot_chord	function	Chord diagram of the transitions between the top most frequent events
plot_communities	function	Pattern communities (Louvain) on the MDS map, with convex hulls and representative patterns
plot_cover	function	Cover of the sequences by the patterns of an MDL summary
plot_dotted_chart	function	Dotted chart: every event a dot at (time, sequence), colored by event
plot_double_tree	function	Word tree in both directions around a pivot event or pattern
plot_embedding	function	2-D map of the patterns from any similarity (or a fused list of similarities)
plot_episode_timeline	function	Minimal occurrences of the most frequent serial episodes on the time axis
plot_explosion	function	Number of all, closed and maximal patterns against the minimum support
plot_gap_distribution	function	Box plots of the time span (or the gaps) of the occurrences of the top patterns
plot_growth_sankey	function	How support splits when frequent sequences grow (support flows of the extension network)
plot_hon	function	Pattern-induced higher-order network (memory nodes around their first-order event)
plot_icicle	function	Icicle plot of the prefix tree (CoreFlow/LifeFlow style)
plot_incidence	function	Pattern–sequence incidence matrix **T** with seriated rows and columns
plot_landscape	function	Pattern landscape: support-weighted density of the MDS map as a terrain with labeled peaks
plot_lattice	function	Hasse diagram of the containment order, one row per pattern length
plot_matrixwave	function	MatrixWave-style sequence of transition matrices between consecutive steps
plot_nbvfs	function	Draw a fitted NBVFS map

Name	Kind	Description
plot_nbvfs_cm	function	Fit and draw the confidence-based MDS map (NBVFS-CM); start_items highlights trees
plot_nbvfs_tm	function	Fit and draw the transaction-based MDS map (NBVFS-TM)
plot_nbvfs_wn	function	Fit and draw the weighted extension network (NBVFS-WN) of a pattern set
plot_overview	function	Six linked views of one mining run: sunburst of the raw sequences, NBVFS-WN, NBVFS-TM, transition graph, sequence index plot with the top pattern, and the suppo
plot_pattern_scatter	function	Patterns as points in the plane of two measures, linked to their parents
plot_patterns	function	Glyph table of the best patterns: events as colored tiles, measures as bars
plot_prediction	function	Prediction fan: the observed events, the matched frequent context, and the ranked continuations
plot_prefix_tree	function	Node-link prefix tree with edges as wide as the number of sequences passing
plot_rules_matrix	function	Grouped matrix of sequential rules: antecedents in columns, consequents in rows
plot_sankey	function	Alluvial diagram of the first steps of the sequences
plot_sequence_frequency	function	The most frequent complete sequences, bar height proportional to their frequency
plot_sequence_index	function	Sequence index plot: one row per sequence, one colored cell per element
plot_sequence_logo	function	Sequence logo of the events around an anchor (or of the first positions)
plot_similarity_matrix	function	Seriated heatmap of a pattern similarity with its dendrogram
plot_state_distribution	function	Chronogram: distribution of the events at every position, with the cross-sectional entropy
plot_sunburst	function	Sequences sunburst: depth on rings, angle proportional to the number of sequences
plot_transition_graph	function	Directly-follows graph: node size = occurrences, edge width = transitions, color = probability
plot_transition_matrix	function	Heatmap of first-order transition counts or probabilities (normalize="row", None)
plot_upset	function	UpSet plot of the supporting sequences of the top patterns
seriate	function	Order of rows/columns that places similar items next to each other (hierarchical clustering)

`tree_data`	function	The tree behind the tree views: a prefix tree of a database or the pattern tree of a pattern set

A.13 `spmviz.interactive`: Interactive figures

Name	Kind	Description
`embedding_figure`	function	Interactive 2-D map of the patterns from a similarity
`icicle_figure`	function	Zoomable icicle of the sequence prefixes
`nbvfs_figure`	function	Interactive version of a fitted NBVFS map (hover shows the measures)
`network_figure`	function	Plotly drawing of any networkx graph (coordinates from x/y or a spring layout)
`sankey_figure`	function	Interactive Sankey diagram of the first steps ("$" = the sequence ended)
`sunburst_figure`	function	Zoomable sunburst of the sequence prefixes (click a wedge to zoom in)
`to_pyvis`	function	A pyvis network (vis.js) of a graph: drag nodes, zoom, hover tooltips
`treemap_figure`	function	Treemap of the sequence prefixes

A.14 `spmviz.io`: Import and export

Name	Kind	Description
`from_igraph`	function	Convert a python-igraph graph back to networkx (name becomes the node key)
`read_graph`	function	Read a graph written by write_graph (GEXF, GraphML, GML, JSON, Cytoscape JSON, CSV)
`read_patterns`	function	Read a pattern set written by write_patterns or by SPMF (#SUP/#SID lines)
`read_spmf`	function	Read a sequence database in SPMF format
`read_xes`	function	Read an XES event log (IEEE 1849) without external dependencies
`to_cytoscape`	function	Cytoscape.js / Cytoscape desktop JSON with node positions
`to_igraph`	function	Convert to a python-igraph graph (node name in the name attribute)
`write_graph`	function	Write a graph with all node and edge attributes (format from the extension)
`write_patterns`	function	Write a pattern set; the format follows the extension
`write_spmf`	function	Write a database in SPMF format

Name	Kind	Description
write_xes	function	Write a database as an XES event log (one trace per sequence, one event per item)

A.15 spmviz.datasets: Data sets

Name	Kind	Description
SPMF_BENCHMARKS	constant	Metadata and download locations of the SPMF benchmark databases
benchmark_table	function	The benchmarks known to fetch_spmf with their published statistics
fetch_spmf	function	Load a benchmark sequence database (downloaded once, then read from the cache)
get_data_home	function	Cache directory of the downloaded benchmarks (created when missing)
load_agrawal_srikant	function	The customer-sequence example of Agrawal and Srikant (1995), *Mining sequential patterns*
load_alarm_log	function	Alarm and operator-action log of an industrial hydrofluoric-acid alkylation plant
load_chess_openings	function	Synthetic chess games drawn from a small opening book of 33 main lines
load_clickstream	function	Click sessions on a web portal of the Dutch Employee Insurance Agency (UWV)
load_melodies	function	Melodies of 13 well-known public-domain songs as note sequences
load_prefixspan_example	function	The four-sequence example database of the PrefixSpan paper (Pei et al., 2001/2004)
make_alarm_cascades	function	Alarm floods made of interleaved cascades of parallel sub-processes
make_customer_journeys	function	Visitor journeys in an online shop, with funnels, loops and abandonment
make_daily_routines	function	Morning-to-evening routines of three personas: early birds, commuters and remote workers
make_planted_patterns	function	Random sequences with planted patterns (ground truth in the attributes)
make_quest	function	IBM Quest-style synthetic sequences of itemsets (Agrawal and Srikant, 1995)

APPENDIX

B



From the NbVFS scripts to spmviz

The scripts published with the NBVFS paper [5] run as a chain of programs that exchange CSV files: `Data_prep.py` converts the log to the SPMF format, runs CM-SPAM with the SPMF Java library and computes parents, confidences and cumulative confidences; `NbVFS-WG.py`, `NbVFS-CM.py` and `NbVFS-TM.py` draw the three views with pyvis, scikit-learn and Matplotlib. Every step has a counterpart in `spmviz`, listed below; no intermediate file is needed, and every intermediate result is an object that can be inspected.

Data preparation and mining

NbVFS scripts	`spmviz`
reading `ind_log.csv` or `Transactional_database.csv`	`SequenceDatabase.from_event_log`; bundled as `datasets.load_alarm_log`, `load_clickstream`
SPMF `Convert_a_sequence_database_to_SPMF_format`	not needed; `io.write_spmf` if SPMF is used
SPMF `CM-SPAM` with minimum support, lengths, maximum gap	`sv.mine(db, min_support, algorithm="cmspam", max_gap=..., max_length=...)`, or `algorithm="spmf"`
parsing `Freqseqs.csv` (`#SUP`, `#SID`)	`io.read_patterns`
`Sequences.csv`: patterns, supports, lengths	`ps.to_frame()`

NbVFS scripts	`spmviz`
`Sequence_attributes.csv`: parent, confidence, `P_conf`, `C_conf`, added event	`ps.parents()`, `ps.confidence()`, `ps.cum_confidence()`, `ps.to_frame()` columns `confidence`, `cum_confidence`, `added`
filter by first event (`filt`, `filt_ev`)	`ps.filter(starts_with=...)` or `NBVFS(..., start_items=[...])`

The three views

NbVFS scripts	`spmviz`
`NbVFS-WG.py`: NetworkX graph of parents and children, pyvis rendering	`NBVFS("wn").fit(ps)`; `.graph_` is the NetworkX graph, `.plot()` draws it, `interactive.to_pyvis` gives the pyvis view
random node colors per length	a fixed, color-blind-safe palette by length [84]
edge value from the child's confidence	`weight="confidence"` (extension confidence, the definition of the paper) or `weight="cumulative"`
`NbVFS-CM.py`: `Adj_matrix_C.csv`, $\max(\mathbf{A}, \mathbf{A}^{\mathsf{T}})$, max-product closure by repeated matrix products	`networks.confidence_matrix`, `networks.transitive_confidence` (one Floyd–Warshall pass)
`NbVFS-TM.py`: `Adj_matrix_T.csv`, Jaccard similarity	`ps.incidence()`, `measures.transaction_similarity(ps, "jaccard")`
`sklearn.manifold.MDS` on $1 - \mathbf{S}$	`embedding.mds(1 - S, "sklearn")`, or the deterministic default `"smacof"` started from classical scaling
Matplotlib scatter with parent edges, highlighted start events	`NBVFS("cm").fit(ps).plot(start_items=[...])`, `NBVFS("tm")`
—	`NBVFS("fused")`, `pattern_communities`, `sublogs` (new)

A complete translation

The whole analysis of the paper for the click-stream, from the raw log to the three views and their files for Gephi, is

```
import spmviz as sv

db = sv.datasets.load_clickstream()
ps = sv.mine(db, min_support=0.01, algorithm="cmspam", max_gap=1,
    max_length=10)
for view in ("wn", "cm", "tm"):
    model = sv.NBVFS(view, start_items=[2] if view == "wn" else
    None).fit(ps)
    model.plot()
    model.write(f"clickstream_{view}.gexf")      # positions, colors,
    sizes for Gephi
```

APPENDIX

C



The example gallery

Every method of the toolbox has at least one runnable example script. The scripts generate or load their own data, explain the method in their comments, and end with a study of the main parameters. The same scripts produce the gallery of the online documentation, the Jupyter notebooks, and every computed figure and code listing of this book. The tables below list them by folder of `examples/`, with the first figure of the book that each script produced.

C.1 First steps: data, mining, constraints, files

Folder `examples/01_basics/`. The methods are described in chapter 2.

Script	Topic	Figure
`plot_01_first_steps.py`	First steps: four sequences of the PrefixSpan paper	2.2
`plot_02_daily_routines.py`	An everyday example: daily routines	2.4
`plot_03_algorithms.py`	Five algorithms, one answer	2.7
`plot_04_constraints.py`	Gaps, spans and time windows	2.8
`plot_05_import_export.py`	Import and export: pandas, SPMF, networkx, Gephi, Cytoscape	—

C.2 Network-based visualization of frequent sequences

Folder `examples/02_nbvfs/`. The methods are described in chapter 4.

Script	Topic	Figure
`plot_01_weighted_network.py`	NBVFS-WN: the weighted extension network	4.1
`plot_02_confidence_mds.py`	NBVFS-CM: confidence-based MDS map	4.4
`plot_03_transaction_mds.py`	NBVFS-TM: transaction-based MDS map	4.6
`plot_04_alarm_log.py`	The industrial alarm log of the paper	4.9
`plot_05_fused_similarity.py`	Beyond the paper: fused similarities and pattern communities	4.11

C.3 Trees: prefix trees, sunbursts, icicles, word trees

Folder `examples/03_trees/`. The methods are described in chapter 5.

Script	Topic	Figure
`plot_01_chess_sunburst.py`	A spectacular sunburst: chess openings	5.2
`plot_02_entropy_tree.py`	Where does the process branch? The continuation-entropy tree	5.5
`plot_03_word_tree.py`	Word trees around a pivot	5.6
`plot_04_contrast_tree.py`	Comparing two groups: contrast sunburst	5.9
`plot_05_lifeflow_icicle.py`	Time-aware icicles (LifeFlow)	5.11

C.4 Flows: Sankey diagrams, transition graphs, higher-order networks

Folder `examples/04_flows/`. The methods are described in chapter 6.

Script	Topic	Figure
`plot_01_sankey.py`	Sankey diagram of web-shop journeys	6.1
`plot_02_transition_graph.py`	Transition graphs, matrices and MatrixWave	6.4
`plot_03_chord.py`	Chord diagram of everyday transitions	6.8
`plot_04_growth_sankey.py`	How support splits when sequences grow	6.10

Script	Topic	Figure
`plot_05_higher_order_network.py`	Pattern-induced higher-order networks: when history matters	6.12

C.5 Maps and matrices of pattern sets

Folder `examples/05_maps/`. The methods are described in chapter 7.

Script	Topic	Figure
`plot_01_incidence_similarity.py`	Matrices of a pattern set	7.1
`plot_02_rules_matrix.py`	Sequential rules as a grouped matrix	7.4
`plot_03_measure_scatter.py`	Patterns in the plane of two measures	7.6
`plot_04_landscape.py`	Pattern landscapes	7.8
`plot_05_lattice_upset.py`	Lattices and UpSet plots	7.10

C.6 Sequences: index plots, logos, arcs, episodes, summaries

Folder `examples/06_sequences/`. The methods are described in chapter 8.

Script	Topic	Figure
`plot_01_index_plot.py`	Sequence index plots with pattern occurrences	8.1
`plot_02_chronogram_frequency.py`	Chronograms and sequence frequency plots	8.3
`plot_03_dotted_chart.py`	Dotted chart of alarm floods	8.5
`plot_04_sequence_logo.py`	Sequence logos	8.7
`plot_05_arc_diagram.py`	Arc diagram of a melody	8.9
`plot_06_episodes.py`	Episodes in one long alarm stream	8.11
`plot_07_mdl_summary.py`	A summary in a few patterns: MDL	8.12

C.7 Applications and benchmarks

Folder `examples/07_applications/`. The methods are described in chapter 10.

Script	Topic	Figure
`plot_01_benchmarks.py`	Benchmark databases of sequential pattern mining	10.1
`plot_02_clickstream_story.py`	Case study: the UWV unemployment-benefit application	10.2
`plot_03_alarm_early_warning.py`	Case study: early warning in alarm management	10.6
`plot_04_interactive.py`	Interactive figures	—

Bibliography

[1] László Bántay and János Abonyi. “Network-based visualisation of frequent sequences”. In: *PLOS ONE* 19.5 (2024), e0301262. DOI: 10.1371/journal.pone.0301262.

[2] Gyula Dörgő and János Abonyi. “Sequence Mining Based Alarm Suppression”. In: *IEEE Access* 6 (2018), pp. 15365–15379. DOI: 10.1109/ACCESS.2018.2797247.

[3] László Bántay, Norbert Sas, Gyula Dörgő, and János Abonyi. “Sequence Compression and Alignment-Based Process Alarm Prediction”. In: *Industrial & Engineering Chemistry Research* 62.27 (2023), pp. 10577–10586. DOI: 10.1021/acs.iecr.3c00935.

[4] László Bántay and János Abonyi. “Machine Learning-Supported Designing of Human–Machine Interfaces”. In: *Applied Sciences* 14.4 (2024), p. 1564. DOI: 10.3390/app14041564.

[5] László Bántay. *NbVFS: Python scripts of the network-based visualisation of frequent sequences*. https://github.com/laszlobantayPE/NbVFS. 2024.

[6] Rakesh Agrawal and Ramakrishnan Srikant. “Mining sequential patterns”. In: *Proceedings of the Eleventh International Conference on Data Engineering*. 1995, pp. 3–14. DOI: 10.1109/ICDE.1995.380415.

[7] Ramakrishnan Srikant and Rakesh Agrawal. “Mining sequential patterns: Generalizations and performance improvements”. In: *Advances in Database Technology – EDBT ’96*. Vol. 1057. Lecture Notes in Computer Science. Springer, 1996, pp. 3–17. DOI: 10.1007/BFb0014140.

[8] Jian Pei et al. “Mining sequential patterns by pattern-growth: The PrefixSpan approach”. In: *IEEE Transactions on Knowledge and Data Engineering* 16.11 (2004), pp. 1424–1440. DOI: 10.1109/TKDE.2004.77.

[9] Mohammed J. Zaki. “SPADE: An efficient algorithm for mining frequent sequences”. In: *Machine Learning* 42.1–2 (2001), pp. 31–60. DOI: 10.1023/A:1007652502315.

[10] Jay Ayres, Jason Flannick, Johannes Gehrke, and Tomi Yiu. “Sequential pattern mining using a bitmap representation”. In: *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2002, pp. 429–435. DOI: 10.1145/775047.775109.

[11] Philippe Fournier-Viger, Antonio Gomariz, Manuel Campos, and Rincy Thomas. "Fast vertical mining of sequential patterns using co-occurrence information". In: *Advances in Knowledge Discovery and Data Mining (PAKDD 2014)*. Vol. 8443. Lecture Notes in Computer Science. Springer, 2014, pp. 40–52. DOI: 10.1007/978-3-319-06608-0_4.

[12] Carl H. Mooney and John F. Roddick. "Sequential pattern mining – approaches and algorithms". In: *ACM Computing Surveys* 45.2 (2013), 19:1–19:39. DOI: 10.1145/2431211.2431218.

[13] Philippe Fournier-Viger, Jerry Chun-Wei Lin, Rage Uday Kiran, Yun Sing Koh, and Rincy Thomas. "A survey of sequential pattern mining". In: *Data Science and Pattern Recognition* 1.1 (2017), pp. 54–77.

[14] Yi Guo, Shunan Guo, Zhuochen Jin, Smiti Kaul, David Gotz, and Nan Cao. "Survey on visual analysis of event sequence data". In: *IEEE Transactions on Visualization and Computer Graphics* 28.12 (2022), pp. 5091–5112. DOI: 10.1109/TVCG.2021.3100413.

[15] Anton Yeshchenko and Jan Mendling. "A survey of approaches for event sequence analysis and visualization". In: *Information Systems* 120 (2024), p. 102283. DOI: 10.1016/j.is.2023.102283.

[16] Wolfgang Jentner and Daniel A. Keim. "Visualization and visual analytic techniques for patterns". In: *High-Utility Pattern Mining: Theory, Algorithms and Applications*. Ed. by Philippe Fournier-Viger, Jerry Chun-Wei Lin, Roger Nkambou, Bay Vo, and Vincent S. Tseng. Springer, 2019, pp. 303–337. DOI: 10.1007/978-3-030-04921-8_12.

[17] Wil M. P. van der Aalst. *Process Mining: Data Science in Action*. 2nd ed. Springer, 2016. DOI: 10.1007/978-3-662-49851-4.

[18] Alexis Gabadinho, Gilbert Ritschard, Nicolas S. Müller, and Matthias Studer. "Analyzing and visualizing state sequences in R with TraMineR". In: *Journal of Statistical Software* 40.4 (2011), pp. 1–37. DOI: 10.18637/jss.v040.i04.

[19] Christian Brzinsky-Fay. "Graphical representation of transitions and sequences". In: *Advances in Sequence Analysis: Theory, Method, Applications*. Ed. by Philippe Blanchard, Felix Bühlmann, and Jacques-Antoine Gauthier. Springer, 2014, pp. 265–284. DOI: 10.1007/978-3-319-04969-4_14.

[20] Ingwer Borg and Patrick J. F. Groenen. *Modern Multidimensional Scaling: Theory and Applications*. 2nd ed. Springer, 2005. DOI: 10.1007/0-387-28981-X.

[21] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, et al. "Array programming with NumPy". In: *Nature* 585 (2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2.

[22] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, et al. "SciPy 1.0: Fundamental algorithms for scientific computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[23] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. "Exploring network structure, dynamics, and function using NetworkX". In: *Proceedings of the 7th Python in Science Conference*. 2008, pp. 11–15.

[24] John D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[25] Chuancong Gao. *prefixspan: The shortest yet efficient implementation of the famous frequent sequential pattern mining algorithm PrefixSpan*. Python package, https://pypi.org/project/prefixspan/. 2019.

[26] Xin Wang, Amin Hosseininasab, Pablo Colunga, Serdar Kadıoğlu, and Willem-Jan van Hoeve. "Seq2Pat: Sequence-to-pattern generation for constraint-based sequential pattern mining". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 36.11 (2022), pp. 12665–12671. DOI: 10.1609/aaai.v36i11.21542.

[27] *GSP-Py: Generalized Sequential Pattern algorithm in Python*. Python package, https://pypi.org/project/gsppy/. 2025.

[28] Philippe Fournier-Viger et al. "The SPMF open-source data mining library version 2". In: *Machine Learning and Knowledge Discovery in Databases (ECML PKDD 2016)*. Vol. 9853. Lecture Notes in Computer Science. Springer, 2016, pp. 36–40. DOI: 10.1007/978-3-319-46131-1_8.

[29] Marcus Dees and Boudewijn F. van Dongen. *BPI Challenge 2016*. 4TU.ResearchData, dataset collection. 2016. DOI: 10.4121/uuid:360795c8-1dd6-4a5b-a443-185001076eab.

[30] Jian Pei et al. "PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth". In: *Proceedings of the 17th International Conference on Data Engineering*. 2001, pp. 215–224. DOI: 10.1109/ICDE.2001.914830.

[31] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. "Discovering frequent closed itemsets for association rules". In: *Database Theory – ICDT'99*. Vol. 1540. Lecture Notes in Computer Science. Springer, 1999, pp. 398–416. DOI: 10.1007/3-540-49257-7_25.

[32] Xifeng Yan, Jiawei Han, and Ramin Afshar. "CloSpan: Mining closed sequential patterns in large datasets". In: *Proceedings of the 2003 SIAM International Conference on Data Mining*. 2003, pp. 166–177. DOI: 10.1137/1.9781611972733.15.

[33] Jianyong Wang and Jiawei Han. "BIDE: Efficient mining of frequent closed sequences". In: *Proceedings of the 20th International Conference on Data Engineering*. 2004, pp. 79–90. DOI: 10.1109/ICDE.2004.1319986.

[34] David Lo, Siau-Cheng Khoo, and Jinyan Li. "Mining and ranking generators of sequential patterns". In: *Proceedings of the 2008 SIAM International Conference on Data Mining*. 2008, pp. 553–564. DOI: 10.1137/1.9781611972788.51.

[35] Philippe Fournier-Viger, Cheng-Wei Wu, Antonio Gomariz, and Vincent S. Tseng. "VMSP: Efficient vertical mining of maximal sequential patterns". In: *Advances in Artificial Intelligence (Canadian AI 2014)*. Vol. 8436. Lecture Notes in Computer Science. Springer, 2014, pp. 83–94. DOI: 10.1007/978-3-319-06483-3_8.

[36] Philippe Fournier-Viger, Antonio Gomariz, Ted Gueniche, Espérance Mwamikazi, and Rincy Thomas. "TKS: Efficient mining of top-k sequential patterns". In: *Advanced Data Mining and Applications (ADMA 2013)*. Vol. 8346. Lecture Notes in Computer Science. Springer, 2013, pp. 109–120. DOI: 10.1007/978-3-642-53914-5_10.

[37] Philippe Fournier-Viger, Ted Gueniche, and Vincent S. Tseng. "Using partially-ordered sequential rules to generate more accurate sequence prediction". In: *Advanced Data Mining and Applications (ADMA 2012)*. Vol. 7713. Lecture Notes in Computer Science. Springer, 2012, pp. 431–442. DOI: 10.1007/978-3-642-35527-1_36.

[38] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. "Discovery of frequent episodes in event sequences". In: *Data Mining and Knowledge Discovery* 1.3 (1997), pp. 259–289. DOI: 10.1023/A:1009748302351.

[39] Anette Eva Fasang and Tim Futing Liao. "Visualizing sequences in the social sciences: Relative frequency sequence plots". In: *Sociological Methods & Research* 43.4 (2014), pp. 643–676. DOI: 10.1177/0049124113506563.

[40] Minseok Song and Wil M. P. van der Aalst. "Supporting process mining by showing events at a glance". In: *Proceedings of the 17th Annual Workshop on Information Technologies and Systems (WITS 2007)*. 2007, pp. 139–145.

[41] Thomas D. Schneider and R. Michael Stephens. "Sequence logos: A new way to display consensus sequences". In: *Nucleic Acids Research* 18.20 (1990), pp. 6097–6100. DOI: 10.1093/nar/18.20.6097.

[42] Martin Wattenberg. "Arc diagrams: Visualizing structure in strings". In: *Proceedings of the IEEE Symposium on Information Visualization 2002*. 2002, pp. 110–116. DOI: 10.1109/INFVIS.2002.1173155.

[43] Jian Zhao, Zhicheng Liu, Mira Dontcheva, Aaron Hertzmann, and Alan Wilson. "MatrixWave: Visual comparison of event sequence data". In: *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. 2015, pp. 259–268. DOI: 10.1145/2702123.2702419.

[44] Martin Krzywinski et al. "Circos: An information aesthetic for comparative genomics". In: *Genome Research* 19.9 (2009), pp. 1639–1645. DOI: 10.1101/gr.092759.109.

[45] Patrick Riehmann, Manfred Hanfler, and Bernd Froehlich. "Interactive Sankey diagrams". In: *IEEE Symposium on Information Visualization 2005*. 2005, pp. 233–240. DOI: 10.1109/INFVIS.2005.1532152.

[46] Adam Perer and Fei Wang. "Frequence: Interactive mining and visualization of temporal frequent event sequences". In: *Proceedings of the 19th International Conference on Intelligent User Interfaces*. 2014, pp. 153–162. DOI: 10.1145/2557500.2557508.

[47] David Gotz and Harry Stavropoulos. "DecisionFlow: Visual analytics for high-dimensional temporal event sequence data". In: *IEEE Transactions on Visualization and Computer Graphics* 20.12 (2014), pp. 1783–1792. DOI: 10.1109/TVCG.2014.2346682.

[48] Sara Di Bartolomeo, Yixuan Zhang, Fangfang Sheng, and Cody Dunne. "Sequence braiding: Visual overviews of temporal event sequences and attributes". In: *IEEE Transactions on Visualization and Computer Graphics* 27.2 (2021), pp. 1353–1363. DOI: 10.1109/TVCG.2020.3030442.

[49] Martin Rosvall, Alcides V. Esquivel, Andrea Lancichinetti, Jevin D. West, and Renaud Lambiotte. "Memory in network flows and its effects on spreading dynamics and community detection". In: *Nature Communications* 5 (2014), p. 4630. DOI: 10.1038/ncomms5630.

[50] Jian Xu, Thanuka L. Wickramarathne, and Nitesh V. Chawla. "Representing higher-order dependencies in networks". In: *Science Advances* 2.5 (2016), e1600028. DOI: 10.1126/sciadv.1600028.

[51] John Stasko and Eugene Zhang. "Focus+context display and navigation techniques for enhancing radial, space-filling hierarchy visualizations". In: *IEEE Symposium on Information Visualization 2000.* 2000, pp. 57–65. DOI: 10.1109/INFVIS.2000.885091.

[52] Joseph B. Kruskal and James M. Landwehr. "Icicle plots: Better displays for hierarchical clustering". In: *The American Statistician* 37.2 (1983), pp. 162–168. DOI: 10.1080/00031305.1983.10482733.

[53] Heidi Lam, Daniel Russell, Diane Tang, and Tamara Munzner. "Session Viewer: Visual exploratory analysis of web session logs". In: *2007 IEEE Symposium on Visual Analytics Science and Technology.* 2007, pp. 147–154. DOI: 10.1109/VAST.2007.4389008.

[54] Krist Wongsuphasawat, John Alexis Guerra Gómez, Catherine Plaisant, Taowei David Wang, Meirav Taieb-Maimon, and Ben Shneiderman. "LifeFlow: Visualizing an overview of event sequences". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems.* 2011, pp. 1747–1756. DOI: 10.1145/1978942.1979196.

[55] Megan Monroe, Rongjian Lan, Hanseung Lee, Catherine Plaisant, and Ben Shneiderman. "Temporal event sequence simplification". In: *IEEE Transactions on Visualization and Computer Graphics* 19.12 (2013), pp. 2227–2236. DOI: 10.1109/TVCG.2013.200.

[56] Zhicheng Liu, Bernard Kerr, Mira Dontcheva, Justin Grover, Matthew Hoffman, and Alan Wilson. "CoreFlow: Extracting and visualizing branching patterns from event sequences". In: *Computer Graphics Forum* 36.3 (2017), pp. 527–538. DOI: 10.1111/cgf.13208.

[57] Martin Wattenberg and Fernanda B. Viégas. "The word tree, an interactive visual concordance". In: *IEEE Transactions on Visualization and Computer Graphics* 14.6 (2008), pp. 1221–1228. DOI: 10.1109/TVCG.2008.172.

[58] Chris Culy and Verena Lyding. "Double Tree: An advanced KWIC visualization for expert users". In: *Proceedings of the 14th International Conference Information Visualisation.* 2010, pp. 98–103. DOI: 10.1109/IV.2010.24.

[59] Cedric Krause, Shivam Agarwal, Michael Burch, and Fabian Beck. "Visually abstracting event sequences as double trees enriched with category-based comparison". In: *Computer Graphics Forum* 42.6 (2023), e14805. DOI: 10.1111/cgf.14805.

[60] Katerina Vrotsou, Jimmy Johansson, and Matthew Cooper. "ActiviTree: Interactive visual exploration of sequences in event-based data using graph similarity". In: *IEEE Transactions on Visualization and Computer Graphics* 15.6 (2009), pp. 945–952. DOI: 10.1109/TVCG.2009.117.

[61] David Gotz, Fei Wang, and Adam Perer. "A methodology for interactive mining and visual analysis of clinical event patterns using electronic health record data". In: *Journal of Biomedical Informatics* 48 (2014), pp. 148–159. DOI: 10.1016/j.jbi.2014.01.007.

[62] Zhicheng Liu, Yang Wang, Mira Dontcheva, Matthew Hoffman, Seth Walker, and Alan Wilson. "Patterns and sequences: Interactive exploration of clickstreams to understand common visitor paths". In: *IEEE Transactions on Visualization and Computer Graphics* 23.1 (2017), pp. 321–330. DOI: 10.1109/TVCG.2016.2598797.

[63] Katerina Vrotsou and Aida Nordman. "Exploratory visual sequence mining based on pattern-growth". In: *IEEE Transactions on Visualization and Computer Graphics* 25.8 (2019), pp. 2597–2610. DOI: 10.1109/TVCG.2018.2848247.

[64] Charles D. Stolper, Adam Perer, and David Gotz. "Progressive visual analytics: User-driven visual exploration of in-progress analytics". In: *IEEE Transactions on Visualization and Computer Graphics* 20.12 (2014), pp. 1653–1662. DOI: 10.1109/TVCG.2014.2346574.

[65] Michael Hahsler and Radoslaw Karpienko. "Visualizing association rules in hierarchical groups". In: *Journal of Business Economics* 87.3 (2017), pp. 317–335. DOI: 10.1007/s11573-016-0822-8.

[66] Michael Hahsler. "arulesViz: Interactive visualization of association rules with R". In: *The R Journal* 9.2 (2017), pp. 163–175. DOI: 10.32614/RJ-2017-047.

[67] Alexander Lex, Nils Gehlenborg, Hendrik Strobelt, Romain Vuillemot, and Hanspeter Pfister. "UpSet: Visualization of intersecting sets". In: *IEEE Transactions on Visualization and Computer Graphics* 20.12 (2014), pp. 1983–1992. DOI: 10.1109/TVCG.2014.2346248.

[68] Jilles Vreeken, Matthijs van Leeuwen, and Arno Siebes. "Krimp: Mining itemsets that compress". In: *Data Mining and Knowledge Discovery* 23.1 (2011), pp. 169–214. DOI: 10.1007/s10618-010-0202-x.

[69] Nikolaj Tatti and Jilles Vreeken. "The long and the short of it: Summarising event sequences with serial episodes". In: *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2012, pp. 462–470. DOI: 10.1145/2339530.2339606.

[70] Apratim Bhattacharyya and Jilles Vreeken. "Efficiently summarising event sequences with rich interleaving patterns". In: *Proceedings of the 2017 SIAM International Conference on Data Mining*. 2017, pp. 795–803. DOI: 10.1137/1.9781611974973.89.

[71] Yuanzhe Chen, Panpan Xu, and Liu Ren. "Sequence synopsis: Optimize visual summary of temporal event data". In: *IEEE Transactions on Visualization and Computer Graphics* 24.1 (2018), pp. 45–55. DOI: 10.1109/TVCG.2017.2745083.

[72] Michael Behrisch, Benjamin Bach, Nathalie Henry Riche, Tobias Schreck, and Jean-Daniel Fekete. "Matrix reordering methods for table and network visualization". In: *Computer Graphics Forum* 35.3 (2016), pp. 693–716. DOI: 10.1111/cgf.12935.

[73] Mohammad Ghoniem, Jean-Daniel Fekete, and Philippe Castagliola. "A comparison of the readability of graphs using node-link and matrix-based representations". In: *IEEE Symposium on Information Visualization*. 2004, pp. 17–24. DOI: 10.1109/INFVIS.2004.1.

[74] Bo Wang et al. "Similarity network fusion for aggregating data types on a genomic scale". In: *Nature Methods* 11.3 (2014), pp. 333–337. DOI: 10.1038/nmeth.2810.

[75] Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. "Fast unfolding of communities in large networks". In: *Journal of Statistical Mechanics: Theory and Experiment* 2008.10 (2008), P10008. DOI: 10.1088/1742-5468/2008/10/P10008.

[76] Guozhu Dong and Jinyan Li. "Efficient mining of emerging patterns: Discovering trends and differences". In: *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1999, pp. 43–52. DOI: 10.1145/312129.312191.

[77] Joseph B. Kruskal. "Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis". In: *Psychometrika* 29.1 (1964), pp. 1–27. DOI: 10.1007/BF02289565.

[78] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. "Gephi: An open source software for exploring and manipulating networks". In: *Proceedings of the International AAAI Conference on Web and Social Media*. Vol. 3. 1. 2009, pp. 361–362. DOI: 10.1609/icwsm.v3i1.13937.

[79] Paul Shannon et al. "Cytoscape: A software environment for integrated models of biomolecular interaction networks". In: *Genome Research* 13.11 (2003), pp. 2498–2504. DOI: 10.1101/gr.1239303.

[80] Alessandro Berti, Sebastiaan van Zelst, and Daniel Schuster. "PM4Py: A process mining library for Python". In: *Software Impacts* 17 (2023), p. 100556. DOI: 10.1016/j.simpa.2023.100556.

[81] Zijian Zheng, Ron Kohavi, and Llew Mason. "Real world performance of association rule algorithms". In: *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2001, pp. 401–406. DOI: 10.1145/502512.502572.

[82] Igor Cadez, David Heckerman, Christopher Meek, Padhraic Smyth, and Steven White. "Model-based clustering and visualization of navigation patterns on a web site". In: *Data Mining and Knowledge Discovery* 7.4 (2003), pp. 399–424. DOI: 10.1023/A:1024992613384.

[83] Panagiotis Papapetrou, George Kollios, Stan Sclaroff, and Dimitrios Gunopulos. "Discovering frequent arrangements of temporal intervals". In: *Proceedings of the Fifth IEEE International Conference on Data Mining*. 2005, pp. 354–361. DOI: 10.1109/ICDM.2005.50.

[84] Bang Wong. "Points of view: Color blindness". In: *Nature Methods* 8.6 (2011), p. 441. DOI: 10.1038/nmeth.1618.



From event logs to frequent sequences to pictures that explain them

Clicks, alarms, operator actions, moves, daily activities: event logs record what happened and in which order. Sequential pattern mining condenses them into the sequences that many cases share, but a list of hundreds of overlapping patterns explains little. This book shows how to see them: how frequent sequences grow out of each other, where a process branches, which patterns belong to the same sub-process, and how groups of cases differ.

It develops the network-based visualization of frequent sequences — a weighted extension network and two maps by multidimensional scaling — and places it among some forty other views from data mining, visual analytics, process mining and sequence analysis: trees, sunbursts and LifeFlow, Sankey diagrams and higher-order networks, matrices, landscapes, lattices, index plots, dotted charts and sequence logos. New views mark the decision points of a process by entropy, contrast two groups, and segment a log into sub-processes.

Every method is implemented in `spmviz`, an open-source Python package with five sequential pattern miners and adapters to SPMF and other libraries; results go to pandas, NetworkX, Gephi and Cytoscape. Each figure in the book is produced by an example script of the package; the same scripts run as Jupyter notebooks.

```
pip install spmviz
```

github.com/ComplexSystemsIntelligence/spmviz