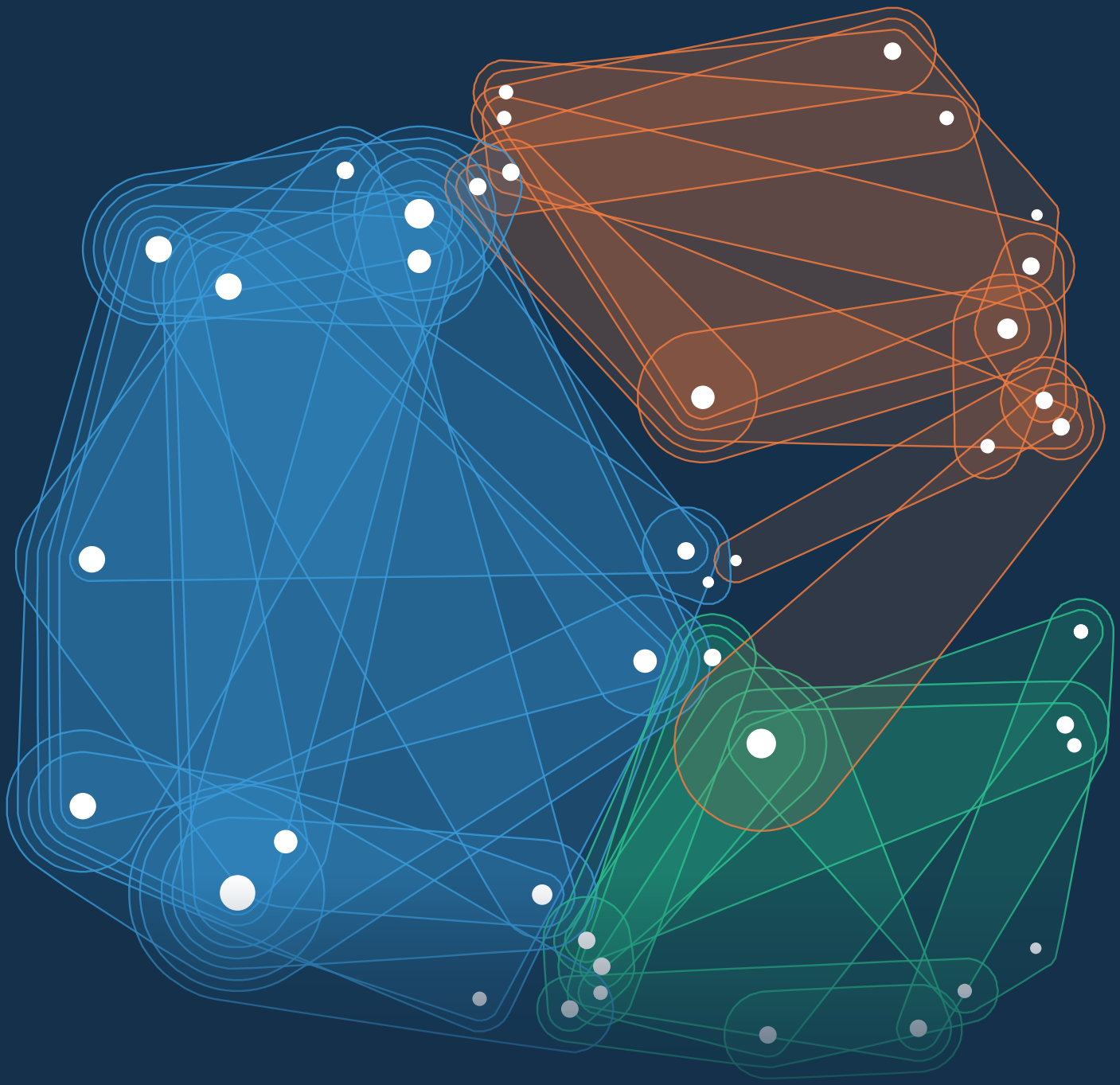


Hypergraph Analysis and Visualization in Python

The `hatvis` toolbox

János Abonyi · Tímea Czvetkó · Attila Ficsor

University of Pannonia, Veszprém



Hypergraph Analysis and Visualization in Python

The `hatvis` toolbox

János Abonyi · Tímea Czvetkó · Attila Ficsor

University of Pannonia, Veszprém

Version 1.0 · November 2026

Hypergraph Analysis and Visualization in Python. The `hatvis` toolbox

János Abonyi, Tímea Czvetkó, and Attila Ficsor, University of Pannonia, Veszprém, Hungary

Version 1.0, November 2026

© 2026 János Abonyi, Tímea Czvetkó, and Attila Ficsor. 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 `hatvis` software described in the book is released under the MIT License.

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

This work was supported by the National Research, Development and Innovation Office of Hungary (NKFIH) through the OTKA K 143482 project *Monitoring complex systems by goal-oriented clustering algorithms*.

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 a hypergraph with three planted communities, laid out by the membership-aware layout of `hatvis`; every hyperedge is a smoothed convex hull in the color of its dominant community.

Preface

This book is about the analysis and visualization of relations that join more than two things at once: a production step that needs an operator, a robot, and a fixture together; a customer requirement that is met only by a combination of engineering solutions; a news article that mentions five themes in the same breath; a set of items bought in one basket. A graph can store such a relation only by breaking it into pairs, and the pairs forget which of them belonged together. A *hypergraph* keeps the group as a whole. The price is that almost every familiar tool of network analysis, from the adjacency matrix to the shortest path, the centrality index, the community, and the drawing, has to be defined again, and usually in more than one way.

The chapters develop these definitions in one notation and, for each of them, show how to compute it with the `hatvis` Python package, what it reveals in a concrete data set, and what it would have missed had the data been reduced to a graph. The emphasis is on methods that a practitioner can apply and interpret: representations and their losses, matrix and node-link views, similarity-based layouts, walks and centralities, communities and partitions, frequent patterns, motifs and null models, combinatorial problems such as covering and acyclicity, quality function deployment, dynamical processes and semi-supervised learning, and the comparison of hypergraphs over time.

Who the book is for. It is written for engineers, data scientists, and graduate students who know basic graph theory and linear algebra and want to work with group interactions without first reading the physics, database, and data-mining literatures separately. The proofs are referenced rather than reproduced; the mathematics needed is that of an undergraduate course in linear algebra and discrete mathematics, together with a little probability for the null models. Every method is stated as an algorithm, summarized in a box at the end of its theory section, and illustrated by a script that the reader can run.

The toolbox and its predecessors. `hatvis` grew out of two visualization tools of the authors' group at the University of Pannonia. *HAT-VIS* [1], written in MATLAB, introduced the similarity-based drawing of hypergraphs: vertices are placed by multidimensional scaling of a weighted Jaccard similarity, the hyperedges are drawn as minimal spanning trees, and the incidence matrix is reordered

by seriation. *HyperGodot* [2], built on the Godot game engine, added interactive force-directed and circular layouts, convex-hull hyperedges with adjustable padding, and the comparison of several versions of a system by groups and line styles. Two applications of the group shaped the analysis part of the package: the hypergraph and network-flow formulation of quality function deployment [3] and the hypergraph-based analysis and design of collaborative manufacturing spaces with s-walk centralities [4]. `hatvis` reimplements all four in Python, places them next to the methods of the wider literature, adds new visualizations (a membership-aware layout, layout quality measures, the s-profile, linked primal–dual views, an interactive HTML viewer), and exchanges data with HyperNetX, XGI, and Hypergraphx through the Hypergraph Interchange Format.

How to run the examples. The package is installed with `pip install hatvis`. The scripts that produce the figures are in the `examples/` directory of the repository, grouped into folders by chapter, and are listed in appendix C. The sphinx-gallery extension renders them as the online gallery of the documentation and converts them into Jupyter notebooks that run in Google Colab without installation. Every number quoted in the text was printed by one of these scripts; when a later version of a dependency changes the last digit, the script, not the book, is authoritative.

Data. The examples use five data sets shipped with the package. Two are taken from the published papers: the electric-vehicle house of quality of [3] and the production example of [2]. The wire-harness manufacturing space of [4] is confidential in its original form; `hatvis` contains a reconstruction with the same resources, the same number of activities, and the same incidences wherever the paper reports them, so that the analysis of chapter 6 can be repeated, but its numbers are not those of the factory. The collection of news articles in chapters 8 and 12 is synthetic: it is generated with planted narratives so that the methods can be checked against a known truth. The fifth set is the small three-hyperedge hypergraph used throughout HAT-VIS.

Acknowledgments. This work was supported by the National Research, Development and Innovation Office of Hungary (NKFIH) through the OTKA K 143482 project *Monitoring complex systems by goal-oriented clustering algorithms*. The authors are grateful to Viktor Sebestyén, co-author of HyperGodot, and to László Nagy, Tamás Ruppert, and Andreas Löcklin, co-authors of the manufacturing-space study, whose ideas run through several chapters. The interchange format and the converters would not exist without the open work of the HyperNetX, XGI, and Hypergraphx teams; the package rests on NumPy, SciPy, pandas, NetworkX, scikit-learn, and Matplotlib [5, 6, 7, 8, 9, 10].

Veszprém, November 2026 János Abonyi, Tímea Czvetkó, and Attila Ficsor

Contents

Preface iii

1 Introduction 1

1.1 Problem and idea 1
1.2 What the hypergraph view buys 2
1.3 The software landscape 4
1.4 Notation 4
1.5 Conventions of the package 5
1.6 A first session 6
1.7 Further reading 7

2 Representations and interchange 9

2.1 Problem and idea 9
2.2 Theory 10
2.3 Interchange 15
2.4 Using it in `hatvis` 17
2.5 Examples 17
2.6 Further reading 21

3 Matrix views 23

3.1 Problem and idea 23
3.2 Theory 24
3.3 Using it in `hatvis` 28
3.4 Examples 28
3.5 Further reading 32

4 Node-link and set views 35

4.1 Problem and idea 35
4.2 Theory 36
4.3 Using it in `hatvis` 40

4.4 Examples . . . 40
4.5 Further reading . . . 44

5 Similarity-based visualization 47

5.1 Problem and idea . . . 47
5.2 Theory . . . 48
5.3 Using it in `hatvis` . . . 52
5.4 Examples . . . 53
5.5 Further reading . . . 56

6 Walks, connectivity, and centrality 57

6.1 Problem and idea . . . 57
6.2 Theory . . . 58
6.3 Using it in `hatvis` . . . 63
6.4 Examples . . . 63
6.5 Further reading . . . 67

7 Clustering and partitioning 69

7.1 Problem and idea . . . 69
7.2 Theory . . . 70
7.3 Using it in `hatvis` . . . 74
7.4 Examples . . . 75
7.5 Further reading . . . 79

8 Frequent patterns and association rules 81

8.1 Problem and idea . . . 81
8.2 Theory . . . 82
8.3 Using it in `hatvis` . . . 86
8.4 Examples . . . 86
8.5 Further reading . . . 89

9 Higher-order structure and null models 91

9.1 Problem and idea . . . 91
9.2 Theory . . . 92
9.3 Using it in `hatvis` . . . 95
9.4 Examples . . . 96
9.5 Further reading . . . 99

10 Combinatorial problem solving 101

10.1 Problem and idea . . . 101
10.2 Theory . . . 102

10.3 Using it in `hatvis` . . . 106
10.4 Examples . . . 107
10.5 Further reading . . . 111

11 Dynamics and learning 113

11.1 Problem and idea . . . 113
11.2 Theory . . . 114
11.3 Using it in `hatvis` . . . 117
11.4 Examples . . . 117
11.5 Further reading . . . 120

12 Temporal and comparative analysis 123

12.1 Problem and idea . . . 123
12.2 Theory . . . 124
12.3 Using it in `hatvis` . . . 126
12.4 Examples . . . 126
12.5 Further reading . . . 129

13 What the hypergraph view buys 131

13.1 The evidence of the examples . . . 131
13.2 From questions to methods . . . 133
13.3 A workflow for a new data set . . . 134
13.4 Limits and outlook . . . 134

A API reference 137

A.1 `hatvis.core` . . . 137
A.2 `hatvis.representations` . . . 137
A.3 `hatvis.io` . . . 138
A.4 `hatvis.similarity` . . . 139
A.5 `hatvis.metrics` . . . 140
A.6 `hatvis.centrality` . . . 141
A.7 `hatvis.spectral` . . . 141
A.8 `hatvis.community` . . . 142
A.9 `hatvis.partition` . . . 142
A.10 `hatvis.patterns` . . . 143
A.11 `hatvis.motifs` . . . 143
A.12 `hatvis.models` . . . 144
A.13 `hatvis.problems` . . . 144
A.14 `hatvis.qfd` . . . 145
A.15 `hatvis.dynamics` . . . 146

A.16 `hatvis.learning` . . . 146
A.17 `hatvis.temporal` . . . 146
A.18 `hatvis.seriation` . . . 147
A.19 `hatvis.layout` . . . 147
A.20 `hatvis.interactive` . . . 148
A.21 `hatvis.datasets` . . . 148
A.22 `hatvis.plotting` . . . 149

B Migrating from HAT-VIS, HyperGodot, and other packages 151

B.1 HAT-VIS . . . 151
B.2 HyperGodot . . . 151
B.3 HyperNetX, XGI, and Hypergraphx . . . 153

C The example gallery 155

C.1 Representations and interchange . . . 155
C.2 Matrix views . . . 155
C.3 Node-link and set views . . . 156
C.4 Similarity-based visualization . . . 156
C.5 Walks, connectivity, and centrality . . . 156
C.6 Clustering and partitioning . . . 157
C.7 Frequent patterns and association rules . . . 157
C.8 Higher-order structure and null models . . . 157
C.9 Combinatorial problem solving . . . 158
C.10 Dynamics and learning . . . 158
C.11 Temporal and comparative analysis . . . 158

Bibliography 161

List of Figures

1.1 Two hypergraphs with the same clique expansion. 2
1.2 The organization of the book. 3
1.3 A first hypergraph and its dual. 7

2.1 Representations of the same hypergraph. 10
2.2 What the clique expansion forgets. 18
2.3 Expansions and line graphs of the production example. 19
2.4 Laplacian spectra of the production example. 20
2.5 Degree and size distributions of the production example. 21

3.1 Seriation of an incidence matrix. 24
3.2 Seriation of the wire-harness incidence matrix. 30
3.3 Biclusters of the wire-harness space. 31
3.4 PAOH view of news articles over three periods. 32
3.5 UpSet view of frequent theme combinations. 33

4.1 Four ways to draw the same hyperedge. 36
4.2 Six layouts of the production example. 41
4.3 Seven hyperedge styles and the vertex filter. 42
4.4 Human and machine tasks of the production example. 43
4.5 Linked primal and dual views. 44
4.6 Circular hub drawings. 45

5.1 The similarity-based visualization pipeline. 48
5.2 The HAT-VIS pipeline on the five-vertex example. 53
5.3 Similarity maps of an electric-car house of quality. 55
5.4 Membership-aware layout on a part of the wire-harness space. 56

6.1 s-walks and s-components. 58
6.2 s-profile of the wire-harness activities. 64
6.3 Vertex s-line graphs of the wire-harness resources. 65

6.4 Critical elements of the wire-harness space. 66
6.5 Spearman correlations of eight hypergraph centralities. 67
6.6 Hyper-core numbers and hyper-coreness. 67

7.1 A split group counted as a hyperedge and as pairs. 70
7.2 Recovery of planted communities as noise increases. 76
7.3 Manufacturing cells of the wire-harness space. 78
7.4 Cut values of four bisections. 78
7.5 Co-clusters of themes and articles. 79

8.1 Patterns as hypergraphs. 82
8.2 The pattern hypergraph of frequent theme combinations. 87
8.3 Inclusion lattice of the climate-science itemsets. 87
8.4 Support, confidence, and lift of association rules. 88
8.5 Statistically validated hyperedges. 89

9.1 A pairwise reshuffle of the configuration model. 92
9.2 h-motif z-scores of the wire-harness space. 96
9.3 Simpliciality of four hypergraphs. 98
9.4 The wire-harness space against two null models. 99

10.1 Hitting set and set cover. 102
10.2 A minimum hitting set of the manufacturing activities. 108
10.3 A database schema and its join tree. 109
10.4 Minimum hyperpath costs of an assembly. 110
10.5 The electric-car house of quality. 111

11.1 Pairwise and group-threshold transmission. 114
11.2 Linear diffusion and nonlinear group consensus. 118
11.3 Threshold contagion with two thresholds and two seeds. 119
11.4 Accuracy of semi-supervised classification. 120

12.1 Temporal and comparative analysis. 124
12.2 Theme activity over three periods. 127
12.3 Difference hypergraph of a redesign. 128

13.1 A workflow for hypergraph analysis. 134

List of Tables

1.1 Hypergraph software related to `hatvis`. . . . 5
1.2 Notation. . . . 6

2.1 Interchange formats and what they preserve. . . . 16

3.1 Seriation quality on the wire-harness space. . . . 29

4.1 Quality of six layouts of the production example. . . . 41

5.1 Membership-aware layout versus MDS and Sammon mapping. . . . 55

6.1 s-component profile of the wire-harness space. . . . 63

7.1 Recovery of planted communities. . . . 76

9.1 Simpliciality, nestedness, and node motifs. . . . 97
9.2 The wire-harness space against two null models. . . . 98

10.1 Three readings of the electric-car house of quality. . . . 110

11.1 Accuracy of semi-supervised classification. . . . 119

13.1 From questions to methods. . . . 133

B.1 HAT-VIS script steps and their `hatvis` counterparts. . . . 152
B.2 HyperGodot features and their `hatvis` counterparts. . . . 152
B.3 Interoperation with HyperNetX, XGI, and Hypergraphx. . . . 153

CHAPTER

1



Introduction

> This chapter explains why group interactions call for a representation of their own, shows on a four-vertex example how much a graph forgets, and maps the rest of the book onto the questions a hypergraph helps to answer. It then places `hatvis` among the existing hypergraph software, fixes the notation used in every chapter, describes the conventions shared by all functions of the package, and ends with a first session that builds, prints, and draws a hypergraph.

1.1 Problem and idea

Network science describes a system by its parts and by the pairs of parts that interact. The description is powerful because the pair is the simplest relation that still has structure: from pairs one obtains paths, distances, degrees, clusters, flows, and the whole apparatus of graph algorithms. Many relations, however, are not pairs. A machining step needs an operator, a machine, and a fixture at the same time; a scientific paper has a set of authors; a chemical reaction consumes a set of reactants; a customer requirement of a product is satisfied by a combination of engineering solutions; a news article mentions a set of themes; a shopping basket holds a set of items. The natural object in each case is a *set* of parts, and the system is a family of such sets. Recent reviews have argued in detail that these higher-order interactions change the structure and the dynamics of complex systems, not only their bookkeeping [11, 12, 13, 14].

A family of subsets of a vertex set is a *hypergraph* [15, 16, 17]. The usual way to analyze it with graph tools is to project it: either onto a graph on the vertices, linking two vertices whenever they appear together in some set (the clique expansion), or onto a bipartite graph whose two kinds of nodes are the vertices and the sets (the star expansion). Both projections are useful, and much of

this book uses them, but the first one loses information that cannot be recovered. Figure 1.1 shows the simplest case. A team of three people $\{a, b, c\}$ and three separate pairs $\{a, b\}, \{b, c\}, \{a, c\}$ have the same clique expansion, a triangle; so do a team of four and the six pairs inside it, or a team of four and four teams of three. Whether the three people ever worked together, whether a requirement needs all three solutions or any two of them, whether a basket contained the three items at once or never more than two, is exactly the information the projection discards.

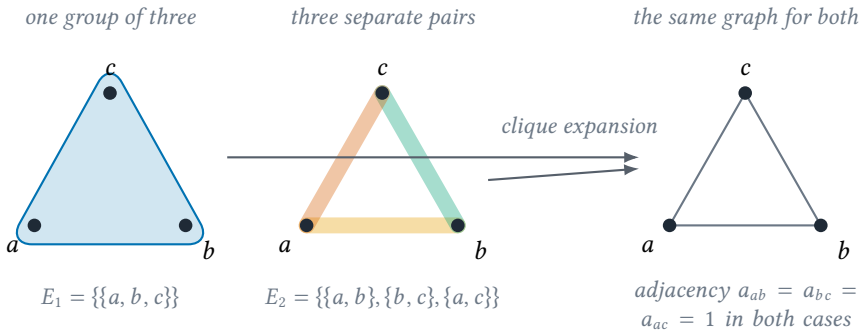


Figure 1.1. Two hypergraphs with the same clique expansion. A single group of three (left) and three separate pairs (middle) produce the same triangle when every hyperedge is replaced by the clique of its members (right). Every pairwise statistic computed on the triangle (degree, clustering coefficient, shortest paths, modularity) is therefore the same for the two systems, although in the first the three vertices always act together and in the second never. Section 2.2.4 gives a general characterization of when a projection loses information.

The idea of the book follows from this example. Instead of choosing one projection and forgetting the hypergraph, we keep the hypergraph as the primary object and derive from it, explicitly and with known losses, whatever representation a question needs: the incidence matrix for seriation and biclustering, the bipartite graph for drawing and for partitioning, the clique expansion for spectral methods, the s-line graphs for walks that require a minimum overlap, the adjacency tensor for centralities that respect the group, the transaction database for pattern mining. Each of these is a view of the same data, and the analysis is honest when the reader knows which view produced which number.

1.2 What the hypergraph view buys

The chapters of the book are organized by the question asked, not by the mathematical object. Figure 1.2 shows the flow: data become a hypergraph, the hypergraph is represented in several ways, the representations feed the views and the analyses, and the analyses feed decisions. Five advantages of the hypergraph view recur throughout and are summarized in chapter 13.

Lossless storage. The incidence matrix **B** stores the groups exactly, with weights, multiplicities, and attributes. Every other representation is computed from it, and the losses of each are known (chapter 2).

Group-aware distances. Two hyperedges are adjacent in the s-line graph only if they share at least s vertices. Walks and centralities parametrized by s distinguish a strongly connected cell from a chain of weak contacts, which no pairwise distance can do (chapter 6).

Group-aware objectives. Hypergraph modularity, cut, and covering objectives count a group once, as a whole, rather than once per pair; a graph objective over-weights large groups quadratically (chapters 7 and 10).

Patterns as hyperedges. A frequent itemset, a validated group, or a motif is a hyperedge of a derived hypergraph; mining, validation, and visualization therefore use the same machinery (chapters 8 and 9).

Nonlinear group effects. Contagion that requires a critical mass in a group, and consensus that depends on the agreement within a group, produce phenomena (hysteresis, bistability, accelerated or slowed convergence) that have no pairwise analogue (chapter 11).

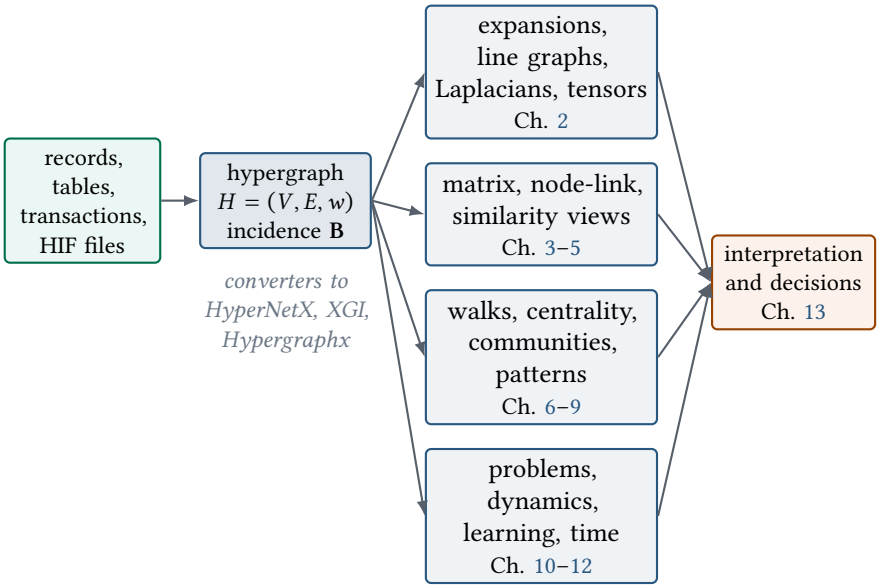


Figure 1.2. The organization of the book. The hypergraph is the primary object; the representations of chapter 2 are derived from it with known losses, and every view and analysis states which representation it uses. Interchange with other packages goes through the Hypergraph Interchange Format and direct converters.

1.3 The software landscape

Hypergraph software has grown quickly in the last five years, and the packages differ more in their purpose than in their data structures. Table 1.1 lists the packages that `hatvis` interoperates with or builds upon.

The *Hypergraph Analysis Toolbox* (HAT) [18], available for MATLAB and Python, was the first general package to base hypergraph analysis on tensor algebra; it contains the hypergraph similarity measures of Surana, Chen, and Rajapakse [19] and methods for controllability and dynamics. *HyperNetX* [20], developed at the Pacific Northwest National Laboratory, implements the s-walk framework of Aksoy et al. [21] and draws hypergraphs as Euler-type diagrams in Matplotlib and in an interactive widget. *XGI* [22] represents higher-order interaction networks with a rich statistics interface, a collection of empirical data sets, and drawing functions for hypergraphs and simplicial complexes. *Hypergraphx* [23] gathers many recent methods of the network-physics community, among them higher-order motifs, statistically validated hyperedges, community models, and dynamical processes. *EasyHypergraph* [24], a module of the EasyGraph framework, aims at speed and memory efficiency for both analysis and hypergraph neural networks, and the DeepHypergraph library accompanies the HGNN$^+$ family of neural models [25]. The Hypergraph Interchange Format (HIF) [26], written jointly by maintainers of several of these packages, defines a JSON schema for undirected and directed hypergraphs and simplicial complexes with attributes on vertices, hyperedges, and incidences; it is the natural way to move data between them.

Two tools of the authors' group are the direct ancestors of `hatvis`. *HAT-VIS* [1] is a MATLAB tool for the similarity-based visualization of hypergraphs; its pipeline (weighted Jaccard similarity, transitive closure, multidimensional scaling, minimal-spanning-tree hyperedges, seriated incidence matrix) is the subject of chapter 5. *HyperGodot* [2] is an interactive viewer built on the Godot engine with force-directed and circular layouts, padded convex hulls, and comparative views of several system variants; its layout and drawing model is the subject of chapter 4.

`hatvis` does not try to replace these packages. Its aims are narrower and complementary: to put the visualization methods of HAT-VIS and HyperGodot, the manufacturing and quality-engineering applications of the group, and the most useful analysis methods of the literature into one consistent, documented, and tested Python API; to make every representation explicit; and to give each method a worked example and a chapter that explains when it should be used.

1.4 Notation

The notation of table 1.2 is used in every chapter, in the docstrings of the package, and in the online documentation. Index i, j always runs over vertices and e, f, g

Table 1.1. Hypergraph software related to `hatvis`. "Interop" lists how `hatvis` exchanges data with the package.

Package	Language	Main focus	Interop
HAT [18]	MATLAB, Python	tensor methods, similarity, controllability	HIF
HyperNetX [20]	Python	s-walks, Euler-type drawings, widget	converter, HIF
XGI [22]	Python	statistics, data sets, drawing	converter, HIF
Hypergraphx [23]	Python	motifs, validation, communities, dynamics	converter, HIF
EasyHypergraph [24]	Python	fast analysis and learning	incidence / edge list
HAT-VIS [1]	MATLAB	similarity-based visualization	reimplemented
HyperGodot [2]	Godot (GDScript)	interactive layouts and comparison	text format, reimplemented

over hyperedges. A vertex is also called a node; the book says *vertex*, the code says `node`, as in NetworkX, HyperNetX, and XGI.

1.5 Conventions of the package

Every function of `hatvis` follows the same small set of rules, so that the results of one method can be passed to another without conversion.

- Functions take the hypergraph as their first argument: `hv.s_betweenness(H, s=2)`, `hv.modularity(H, partition)`.
- Results at the level of vertices or hyperedges are `pandas.Series` indexed by the labels; tables are `pandas.DataFrame`; matrices are NumPy arrays aligned with `H.nodes` and `H.edges`, or SciPy sparse arrays where this is stated.
- Partitions are `pandas.Series` of consecutive integers $0, \dots, k-1$ indexed by the vertex labels.
- Randomness is controlled by a `random_state` argument (`None`, an integer, or a NumPy `Generator`); all examples use `random_state=0`.
- Layout functions are called `layout_*` and return a dictionary from vertex label to a NumPy array of two coordinates; every drawing function accepts such a dictionary as `pos`.
- Plotting functions are called `plot_*` or, for node-link drawings, `draw_*`; they take an optional Matplotlib `ax`, return it, and never call `plt.show()`. Categorical colors come from the Okabe–Ito palette [27], which remains distinguishable for the common forms of color-vision deficiency.
- Optional dependencies (HyperNetX, XGI, Hypergraphx, openpyxl) are imported only inside the functions that need them.

Table 1.2. Notation used throughout the book and in the docstrings of `hatvis`.

Symbol	Meaning	Code
$H = (V, E, w)$	hypergraph: vertices, hyperedges, weights	`Hypergraph`
$n = \lvert V\rvert$, $m = \lvert E\rvert$	numbers of vertices and hyperedges	`H.n_nodes`, `H.n_edges`
$\mathbf{B} = [b_{ie}] \in \mathbb{R}^{n\times m}$	incidence matrix, $b_{ie} > 0$ iff $v_i \in e$	`H.incidence`
w_e, $\mathbf{W} = \operatorname{diag}(w_e)$	hyperedge weights	`H.weights`
$\lvert e\rvert$	size (cardinality) of hyperedge e	`H.edge_sizes()`
$d_i = \sum_e w_e b_{ie}$	(weighted) degree of vertex i	`H.degrees()`
$\mathbf{D}_v$, $\mathbf{D}_e$	diagonal degree and size matrices	
H^*	dual hypergraph, incidence $\mathbf{B}^\mathsf{T}$	`H.dual()`
$L_s(H)$	s-line graph: $e \sim f \iff \lvert e \cap f\rvert \geq s$	`s_line_graph(H, s)`
$\mathbf{A} = \mathbf{B}\mathbf{W}\mathbf{B}^\mathsf{T} - \mathbf{D}_v$	clique-expansion adjacency	`clique_adjacency(H)`
$\mathbf{L}$	a hypergraph Laplacian	`laplacian(H, kind)`
$\mathbf{S} = [s_{ij}]$, $\mathbf{\Delta} = \mathbf{1} - \mathbf{S}$	vertex similarity and dissimilarity	`vertex_similarity(H)`
$\mathbf{y}_i \in \mathbb{R}^2$, $\mathbf{Y}$	layout coordinates	dict `label -> (x, y)`
$\mathcal{T} = \{T_1, \dots, T_N\}$	transaction database	`hypergraph_to_transactions`
$\operatorname{supp}(X)$	relative support of itemset X	`frequent_itemsets`
$\boldsymbol{\sigma}$	partition (community labels)	`pandas.Series` of `int`

- Hypergraphs are immutable in spirit: every transformation (restriction, dual, collapse, filtering, α-cut) returns a new object and leaves the original unchanged.

The package depends on NumPy, SciPy, pandas, NetworkX, scikit-learn, and Matplotlib. The integer programs of chapter 10 use the HiGHS solver [28] through `scipy.optimize.milp`.

1.6 A first session

The smallest example of the HAT-VIS paper [1] has five vertices and three hyperedges of three vertices each. It is built from a dictionary that maps hyperedge labels to member lists; lists of member sets, incidence matrices, and long-format data frames are accepted as well (section 2.4).

```
H = hv.Hypergraph({"e1": ["v1", "v2", "v3"], "e2": ["v1", "v2", "v4"],
    "e3": ["v3", "v4", "v5"]})
print(H.to_dataframe().astype(int))
print(H.degrees().to_dict(), H.edge_sizes().to_dict())
D = H.dual()
```

```
print(D.edge_dict())
```

The incidence matrix has a row per vertex and a column per hyperedge. The degrees are $d_{v_1} = \cdots = d_{v_4} = 2$ and $d_{v_5} = 1$; all three hyperedges have size 3. The dual hypergraph exchanges the two roles: its vertices are e_1, e_2, e_3, and every original vertex becomes a hyperedge listing the hyperedges that contain it, so v_1 becomes $\{e_1, e_2\}$ and v_5 the singleton $\{e_3\}$. The two views of fig. 1.3 are complete descriptions of the same object: the matrix shows every incidence explicitly and scales to thousands of rows, while the drawing shows at a glance which groups overlap. Neither is a projection; both can be computed back into the other. Already this small example has a feature that the following chapters return to: v_1 and v_2 belong to exactly the same hyperedges, so no analysis based on incidences alone can tell them apart, and a similarity-based layout places them on top of each other (chapter 5).

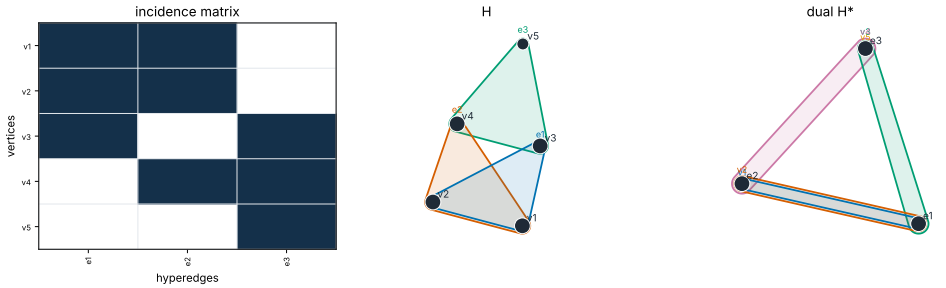


Figure 1.3. A first hypergraph and its dual. Left: the incidence matrix of the five-vertex example of HAT-VIS. Middle: the hypergraph drawn with padded convex hulls on the force-directed layout of HyperGodot. Right: the dual hypergraph, whose vertices are the original hyperedges; v_1 and v_2 become two identical hyperedges $\{e_1, e_2\}$ and v_5 a singleton.

1.7 Further reading

Berge's monographs [15, 16] remain the standard reference for the combinatorics of hypergraphs, and Bretto [17] gives a modern introduction. The review of Battiston et al. [11] surveys the structure and dynamics of networks beyond pairwise interactions; Bick et al. [13] compare hypergraphs, simplicial complexes, and other higher-order models from a mathematical viewpoint, and Torres et al. [14] discuss when each representation of a complex system is appropriate. Lee et al. [29] survey the structural patterns observed in real-world hypergraphs and the tools and generators used to find them. For visualization, the surveys of Alsallakh et al. on set visualization [30] and of Fischer et al. on hypergraph visualization [31] are the best starting points.

CHAPTER

2



Representations and interchange

A hypergraph can be stored, drawn, and computed with in many forms: as an incidence matrix, as a bipartite graph, as a graph of co-occurring vertices, as a graph of overlapping hyperedges, as a tensor, or as a file in one of several exchange formats. This chapter defines these representations in one notation, states which of them are lossless and what the others forget, introduces the variants of the basic model (weighted, fuzzy, directed, and temporal hypergraphs), and describes how `hatvis` reads, writes, and converts all of them. Every later chapter names the representation it works on, so this chapter is the reference for the rest of the book.

2.1 Problem and idea

The same family of groups admits very different encodings, and each encoding makes some questions easy and others impossible. The incidence matrix answers "which groups contain this vertex" with a row lookup and supports seriation, biclustering, and every linear-algebraic method; the bipartite graph makes a hypergraph accessible to graph drawing and to partitioning software; the graph of co-occurring vertices makes spectral clustering and diffusion straightforward; the graph of overlapping groups lets one walk from group to group; the adjacency tensor keeps every group as a single multilinear term. Figure 2.1 shows several of them for the small example of section 1.6.

The difficulty is that some of these encodings are not invertible. Two hypergraphs with the same clique expansion are indistinguishable to any method that uses only that expansion (fig. 1.1); two vertices with the same memberships are indistinguishable to any method that uses only the incidences; the line graph forgets

which vertices produced an overlap. The idea of this chapter is to keep a single lossless object, the incidence matrix with its attributes, to compute every other representation from it on demand, and to state for each representation exactly what it forgets. A method whose answer differs between two hypergraphs that share a representation cannot be computed from that representation, and a method that uses a lossy representation must be read with its loss in mind.

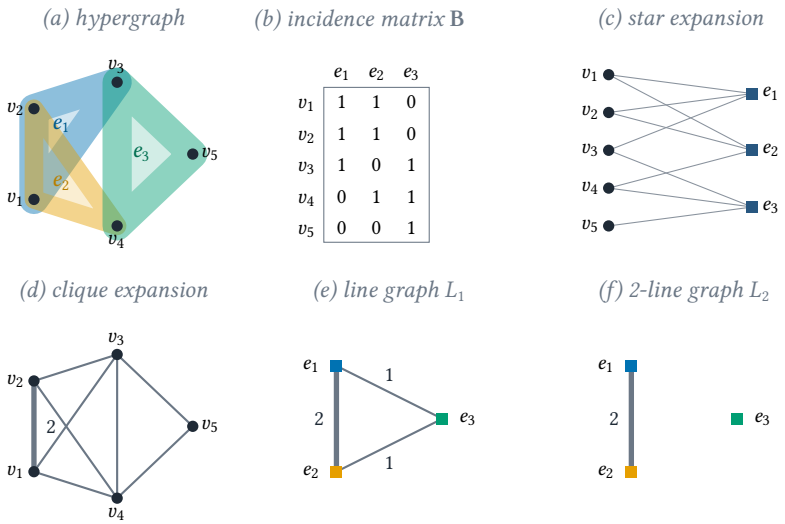


Figure 2.1. Representations of the same hypergraph. The drawing (a), the incidence matrix (b), and the star expansion (c) are lossless: each can be converted into the others. The clique expansion (d) records only which pairs of vertices meet, with the number of shared hyperedges as weight (v_1v_2 meet twice); the line graph (e) records only which hyperedges overlap and by how much. Raising the overlap threshold to $s = 2$ (f) keeps only the pair e_1e_2, which share v_1 and v_2.

2.2 Theory

2.2.1 Hypergraphs and their variants

A *hypergraph* $H = (V, E)$ consists of a finite vertex set $V = \{v_1, \dots, v_n\}$ and a family $E = (e_1, \dots, e_m)$ of subsets of V, the *hyperedges* [16]. The word family matters: the same subset may occur more than once (a multi-hypergraph), and hyperedges may be empty. The *size* of a hyperedge is $|e|$, the *degree* of a vertex is the number of hyperedges that contain it, the *rank* and *co-rank* of H are the largest and smallest hyperedge sizes, and H is *k-uniform* when all hyperedges have size k; a graph is a 2-uniform hypergraph. A hypergraph is *simple* (a clutter, or Sperner family) when no hyperedge is contained in another; the maximal hyperedges of any hypergraph, its *toplexes*, form a simple hypergraph.

The *incidence matrix* $\mathbf{B} \in \mathbb{R}^{n \times m}$ has $b_{ie} = 1$ if $v_i \in e$ and 0 otherwise. Everything else can be added to this matrix without changing its shape:

- *hyperedge weights* $w_e > 0$, collected in $\mathbf{W} = \operatorname{diag}(w_e)$, express the duration, cost, frequency, or importance of a group; the weighted degree is $d_i = \sum_e w_e b_{ie}$;
- *real-valued incidences* $b_{ie} \in (0, 1]$ express graded membership. Lee-Kwang and Lee [32] call such an object a *fuzzy hypergraph* and define its crisp α-cut H_α by keeping the incidences with $b_{ie} \geq \alpha$; the family $\{H_\alpha\}$ is nested and shows which groups persist as the membership requirement is tightened. The same real-valued matrix also encodes a vertex-weighted incidence, as in the quality-function-deployment hypergraphs of section 10.2.5, where b_{ie} is the strength of the relationship between a requirement and a solution;
- *attributes* of vertices and hyperedges (a type, a group, a time stamp, a name) are kept in two tables indexed by the labels.

Three further variants are needed in the applications. A *directed hypergraph* [33, 34] has hyperarcs $e = (T(e), H(e))$ with a tail set and a head set; it represents AND-type dependencies, such as an assembly step that needs all its inputs, a chemical reaction, a Horn clause, or an association rule $X \Rightarrow Y$. It is stored as two incidence matrices $\mathbf{B}^{\text{tail}}, \mathbf{B}^{\text{head}}$ of the same shape. A *temporal hypergraph* attaches a time stamp t_e to every hyperedge; its snapshots $H_t = (V, \{e : t_e = t\})$ and windows $H_{[t_0,t_1)}$ are ordinary hypergraphs (chapter 12). Finally the *dual hypergraph* H^* exchanges the roles of vertices and hyperedges: its incidence matrix is $\mathbf{B}^\mathsf{T}$, so the hyperedge of H^* belonging to v_i is the set of hyperedges of H that contain v_i, and $(H^*)^* = H$. The dual is lossless; every vertex-level method becomes a hyperedge-level method when applied to H^*, which is how `hatvis` computes most hyperedge centralities and similarities.

Two subhypergraphs are used constantly. The *partial hypergraph* on a subset $F \subseteq E$ keeps those hyperedges and all vertices; the *induced subhypergraph* on $U \subseteq V$ cuts every hyperedge to $e \cap U$. Restricting to a production line, to a time window, or to the hyperedges of one automation group are examples of the first; restricting to the resources of one cell is an example of the second.

2.2.2 Matrices derived from the incidence matrix

Two products of the incidence matrix with its transpose count overlaps:

$$\mathbf{C} = \mathbf{B}\mathbf{B}^\mathsf{T}, \quad c_{ij} = |\{e : v_i, v_j \in e\}|, \qquad \mathbf{K} = \mathbf{B}^\mathsf{T}\mathbf{B}, \quad k_{ef} = |e \cap f|, \tag{2.1}$$

for a 0/1 matrix $\mathbf{B}$. The *co-occurrence matrix* $\mathbf{C}$ has the degrees on its diagonal and the number of shared hyperedges off it; the *intersection matrix* $\mathbf{K}$ has the sizes on its diagonal and the overlaps off it. Both are the basis of the similarity measures of chapter 5, of the line graphs below, and of the pattern statistics of chapter 8.

The weighted adjacency of the clique expansion generalizes $\mathbf{C}$:

$$a_{ij} = \sum_{e \ni v_i, v_j} w_e \, c(e) \quad (i \neq j), \qquad c(e) \in \left\{ 1, \tfrac{1}{|e|}, \tfrac{1}{|e|-1} \right\}, \tag{2.2}$$

and $a_{ii} = 0$. With $c(e) = 1$ the weight counts the shared hyperedges (or sums their weights), so $\mathbf{A} = \mathbf{B}\mathbf{W}\mathbf{B}^{\mathsf{T}} - \mathbf{D}_v$ for a 0/1 incidence; with $c(e) = 1/|e|$ large groups contribute less per pair, the normalization underlying the Laplacian of Zhou et al. [35]; with $c(e) = 1/(|e| - 1)$ every vertex receives exactly w_e in total from each of its hyperedges, so the weighted degree of the expansion equals the weighted hypergraph degree. Agarwal et al. [36] showed that many Laplacians proposed for hypergraph learning reduce to Laplacians of the clique or the star expansion with one of these weightings, which is why the choice of $c(e)$ matters more than the name of the method.

2.2.3 Expansions and line graphs

The *clique expansion* (or 2-section, primal graph) $[H]_2$ has the vertex set V and an edge between two vertices whenever they share a hyperedge, weighted by (2.2). The *star expansion* (incidence graph, Levi graph, or König representation) is the bipartite graph with vertex set $V \cup E$ and an edge $\{v_i, e\}$ for every incidence $b_{ie} \neq 0$, weighted by b_{ie} [17]. It is lossless, and every graph algorithm that respects the bipartition (drawing in two layers, bipartite matching, bipartite community detection, minimum cuts that may separate a vertex from a hyperedge) applies to it directly.

Aksoy et al. [21] introduced a family of graphs that interpolate between "any contact" and "strong contact". The *s-line graph* $L_s(H)$ has the hyperedges of size at least s as its vertices and an edge between e and f whenever

$$|e \cap f| \;\geq\; s, \tag{2.3}$$

with weight $|e \cap f| = k_{ef}$. For $s = 1$ it is the ordinary line graph (the intersection graph of the hyperedges); as s grows, only hyperedges with substantial overlap remain connected, and the components of $L_s(H)$ split. The vertex version is the s-line graph of the dual: two vertices are adjacent when they share at least s hyperedges, i.e. when $c_{ij} \geq s$. A walk in $L_s(H)$ is an *s-walk* of the hypergraph, a sequence of hyperedges in which consecutive ones share at least s vertices; distances, components, and centralities defined on s-walks are the subject of chapter 6. Figure 2.1(e) and (f) show L_1 and L_2 of the small example.

2.2.4 What is lost

Three kinds of loss occur.

Algorithm 2.1 Conformality test

Input: hypergraph H
Output: whether H is conformal
build the clique expansion $[H]_2$ (2.2);
foreach *maximal clique* Q *of* $[H]_2$ *(Bron–Kerbosch enumeration)* **do**
  if $|Q| > 1$ *and no hyperedge* e *satisfies* $Q \subseteq e$ **then return** *false*;
return *true*

Clique expansion. The clique expansion cannot distinguish a hyperedge from the set of all pairs inside it, nor a hyperedge from several smaller hyperedges that cover the same pairs (fig. 1.1). The cases in which it loses nothing essential are characterized by *conformality*. A hypergraph is conformal when every maximal clique of its clique expansion is contained in some hyperedge [16]. For a simple conformal hypergraph the maximal cliques of $[H]_2$ are exactly the hyperedges, so H is recovered from $[H]_2$ by listing its maximal cliques; for a conformal hypergraph that is not simple, the toplexes are recovered and the hyperedges contained in others are lost; for a non-conformal hypergraph some maximal clique of $[H]_2$ (in fig. 1.1, the triangle $\{a, b, c\}$ of the three pairs) is not a group at all, and the expansion suggests an interaction that never happened. Gilmore's criterion gives a direct test: H is conformal if and only if for any three hyperedges e, f, g the set $(e \cap f) \cup (f \cap g) \cup (e \cap g)$ is contained in some hyperedge [16]. `hatvis` tests conformality by enumerating the maximal cliques of the expansion (algorithm 2.1), which is fast for the sparse expansions met in practice.

Line graphs. The s-line graph keeps the overlaps but forgets the vertices that produced them: in fig. 2.1(e) one cannot tell which two vertices e_1 and e_2 share, or whether e_3 meets them in the same vertex. A single line graph also forgets the hyperedges smaller than s. The family $\{L_s(H)\}_{s\geq 1}$ together with the sizes carries much more information, and it is this family, not a single member, that the analysis of chapter 6 uses.

Incidences. Even the lossless incidence matrix cannot distinguish two vertices with identical rows, i.e. with the same memberships. Such vertices are *equivalent*: every method that is a function of $\mathbf{B}$ treats them identically, every similarity between them is maximal, and a similarity-based layout places them on top of each other. `H.collapse_nodes()` merges each class into one representative and records the class sizes, and `H.collapse()` merges identical hyperedges and adds their multiplicities to the weights. Both reductions are lossless for methods that take weights into account and simplify the drawings considerably.

2.2.5 Laplacians

Spectral methods (clustering, embedding, diffusion, consensus, label spreading) need a Laplacian, and the hypergraph literature offers several. With $\mathbf{D}_v = \operatorname{diag}(d_i)$ and $\mathbf{D}_e = \operatorname{diag}(|e|)$, `hatvis` provides

$$\mathbf{L}_{\text{Zhou}} = \mathbf{I} - \mathbf{D}_v^{-1/2}\,\mathbf{B}\mathbf{W}\mathbf{D}_e^{-1}\mathbf{B}^{\mathsf{T}}\,\mathbf{D}_v^{-1/2}, \tag{2.4}$$

$$\mathbf{L}_{\text{rw}} = \mathbf{I} - \mathbf{D}_v^{-1}\,\mathbf{B}\mathbf{W}\mathbf{D}_e^{-1}\mathbf{B}^{\mathsf{T}}, \tag{2.5}$$

$$\mathbf{L}_{\text{Bolla}} = \mathbf{D}_v - \mathbf{B}\mathbf{W}\mathbf{D}_e^{-1}\mathbf{B}^{\mathsf{T}}, \tag{2.6}$$

$$\mathbf{L}_{\text{clique}} = \operatorname{diag}(\mathbf{A}\mathbf{1}) - \mathbf{A}, \qquad \mathbf{A} \text{ from (2.2) with } c(e) = 1. \tag{2.7}$$

The normalized Laplacian (2.4) was introduced by Zhou, Huang, and Schölkopf [35]; (2.5) is its random-walk form, $\mathbf{I} - \mathbf{P}$ with the transition matrix $\mathbf{P}$ of the walk that leaves a vertex through one of its hyperedges chosen with probability proportional to w_e and then moves to a uniformly chosen member of that hyperedge (section 6.2.4); (2.6) is the combinatorial Laplacian of Bolla [37]; and (2.7) is the Laplacian of the clique expansion weighted by the number of shared hyperedges, studied by Rodríguez [38].

All four are symmetric positive semidefinite (the random-walk form is similar to the normalized one) and have a zero eigenvalue for every connected component of the hypergraph, so the number of zero eigenvalues counts the components and the second smallest eigenvalue, the *algebraic connectivity*, is positive exactly when the hypergraph is connected. Their spectra nevertheless differ in a way that is worth knowing. Writing $\mathbf{L}_{\text{Zhou}} = \mathbf{I} - \mathbf{X}\mathbf{X}^{\mathsf{T}}$ with $\mathbf{X} = \mathbf{D}_v^{-1/2}\mathbf{B}\mathbf{W}^{1/2}\mathbf{D}_e^{-1/2} \in \mathbb{R}^{n\times m}$ shows that its eigenvalues lie in $[0, 1]$ and that the matrix $\mathbf{X}\mathbf{X}^{\mathsf{T}}$ has rank at most m, so *at least $n - m$ eigenvalues are exactly 1*: when there are more vertices than hyperedges, a large part of the spectrum carries no structural information at all. The combinatorial Laplacians (2.6) and (2.7) have no such plateau, and their largest eigenvalues grow with the degrees. Agarwal et al. [36] discuss the equivalence of these operators with Laplacians of the expansions; Hayashi et al. [39] and Chitra and Raphael [40] generalize the random walk to edge-dependent vertex weights, and Lucas et al. [41] combine the Laplacians of the uniform layers into a multiorder Laplacian.

2.2.6 Tensors and layers

A k-uniform hypergraph is naturally a symmetric tensor of order k. The *adjacency tensor* of Cooper and Dutle [42] is

$$\mathcal{A}_{i_1 i_2 \dots i_k} = \frac{w_e}{(k-1)!} \quad \text{if } \{v_{i_1}, \dots, v_{i_k}\} = e \in E, \qquad 0 \text{ otherwise}, \tag{2.8}$$

and the tensor–vector product

$$(\mathcal{A}\mathbf{x}^{k-1})_i = \sum_{e \ni v_i} w_e \prod_{v_j \in e \setminus \{v_i\}} x_j \tag{2.9}$$

In a nutshell Representations

- Lossless: incidence matrix $\mathbf{B}$ (with weights $\mathbf{W}$, incidence values, and attribute tables), star expansion, dual H^*. `hatvis` stores $\mathbf{B}$ as a sparse matrix and computes every other form from it.
- Lossy: clique expansion (recovers the toplexes only if H is conformal), s-line graphs (forget the vertices of the overlaps), any statistic of the incidences (cannot separate vertices with equal memberships).
- Overlap counts: $\mathbf{B}\mathbf{B}^\mathsf{T}$ (shared hyperedges) and $\mathbf{B}^\mathsf{T}\mathbf{B}$ (shared vertices) underlie similarities, line graphs, and pattern statistics.
- Laplacians: Zhou normalized and random walk (eigenvalues in $[0, 1]$, at least $n - m$ equal to 1), Bolla, clique (Rodríguez); zero eigenvalues count components.
- Tensors: adjacency tensor of uniform hypergraphs and the tensor–vector product (2.9), valid for any hypergraph.
- Variants: multi-, weighted, fuzzy (α-cuts), directed (tail/head), temporal (snapshots, windows).

is the building block of the tensor eigenvector centralities of section 6.2.5. The right-hand side of (2.9) is defined for any hypergraph, uniform or not, and `hatvis` evaluates it from the incidence lists without ever forming the n^k entries of the tensor; the product over the other members is what makes these methods group-aware, since a vertex receives credit from a hyperedge only if *all* its co-members are important. HAT [18] builds its analysis on this representation, including the Laplacian tensor $\mathcal{L} = \mathcal{D} - \mathcal{A}$, which `hatvis` also provides for small uniform hypergraphs.

A non-uniform hypergraph can be split into its *uniform layers* $H^{(k)} = (V, \{e : |e| = k\})$, a multilayer view in which pairwise and higher-order interactions are kept apart. Its opposite is the *simplicial closure*, which adds every non-empty subset of every hyperedge and turns H into an abstract simplicial complex; it is exponential in the hyperedge size and is used here only to construct examples and to measure how far a hypergraph is from a simplicial complex (section 9.2.4).

2.3 Interchange

A representation is only useful if the data can get into it. `hatvis` reads and writes five families of formats and converts directly to and from three Python packages; table 2.1 lists what each of them preserves.

Hypergraph Interchange Format. HIF [26] is a JSON schema agreed by maintainers of HyperNetX, XGI, Hypergraphx, HAT, and other packages. A HIF document has a list of `incidences`, each a record with an `edge` and a `node` identifier

and optional `weight`, `direction`, and `attrs`; optional lists of `nodes` and `edges` carry their own weights and attributes; the field `network-type` distinguishes undirected and directed hypergraphs and abstract simplicial complexes. `hatvis` writes hyperedge weights to the edge records, incidence values to the incidence records, the attribute tables to `attrs`, and the tail/head sets of a directed hypergraph to the `direction` field, so a round trip through HIF gives back an identical object.

HyperGodot text format. HyperGodot [2] reads one hyperedge per line, the vertex labels separated by spaces, followed by optional tags such as `#WEIGHT: 15` and `#GROUP: H`. The weight becomes the hyperedge weight; every other tag becomes a column of the hyperedge attribute table, which is how the comparative views of section 4.2.5 find their groups. Files written by `hatvis` open in HyperGodot unchanged.

Pattern-mining output. The SPMF library [43] writes frequent itemsets as lines `1 4 7 #SUP: 25` and association rules as `1 2 ==> 5 #SUP: 3 #CONF: 0.75`. `hatvis` reads the first as a hypergraph whose hyperedges are the itemsets weighted by their support and the second as a directed hypergraph whose hyperarcs are the rules, so the output of any external miner can be visualized and analyzed with the methods of chapter 8.

Tables. Incidence data usually arrive as a long table with one row per membership (a person and a project, an item and a basket, a resource and an activity) or as a wide matrix in a spreadsheet. `Hypergraph.from_dataframe` reads the first, `read_incidence_csv` and `read_incidence_excel` the second; the house-of-quality matrix of section 5.2.6 was read from the Excel file of the HAT-VIS distribution in this way.

Python packages. The converters `to_hypernetx`/`from_hypernetx`, `to_xgi`/`from_xgi`, and `to_hypergraphx`/`from_hypergraphx` create the native objects of HyperNetX [20], XGI [22], and Hypergraphx [23] with the hyperedge weights as attributes, so a method that exists only in one of these packages can be applied to a `hatvis` hypergraph and its result brought back by label. The packages are optional dependencies and are imported only when a converter is called.

Table 2.1. Interchange formats and what they preserve in `hatvis`. "Labels" means hyperedge labels; vertex labels are preserved by all.

Format	Labels	Weights	Incidence values	Attributes	Directed
HIF JSON	yes	yes	yes	yes	yes
HyperGodot text	generated	yes	no	tags	no
SPMF itemsets / rules	generated	support	no	confidence	rules
CSV / Excel incidence	yes	column	yes	no	no
HyperNetX / XGI	yes	yes	no	partly	no
Hypergraphx	no	yes	no	no	no

2.4 Using it in `hatvis`

`Hypergraph(edges, *, nodes=None, weights=None, edge_labels=None, node_data=None, edge_data=None, name=None)`

`edges` a list of member lists (labelled `e0`, `e1`, …), a dictionary `label -> members`, or a dictionary `label -> {vertex: value}` for real-valued incidences.

`nodes` vertex order; vertices listed here but in no hyperedge are kept as isolated vertices.

`weights` hyperedge weights w_e (default 1).

`node_data`, `edge_data` attribute tables indexed by the labels.

Constructors `Hypergraph.from_incidence(B, nodes, edges)` (array, sparse matrix, or labelled data frame), `Hypergraph.from_dataframe(df, node_col, edge_col, value_col)`, `DirectedHypergraph(arcs)`, `TemporalHypergraph(edges, times)`.

Access `incidence`, `nodes`, `edges`, `weights`, `members(e)`, `memberships(v)`, `edge_dict()`, `node_dict()`, `to_dataframe()`, `summary()`.

Transformations `dual()`, `restrict_nodes`, `restrict_edges`, `filter_edges(min_size, max_size, min_weight, predicate)`, `alpha_cut(alpha)`, `collapse()`, `collapse_nodes()`, `toplexes()`, `relabel`, `add_edges`; all return new objects.

The representations of section 2.2 are plain functions of the hypergraph: `clique_adjacency(H, weighting)` and `clique_expansion`, `star_expansion`, `line_graph(H, s)` and `s_line_graph(H, s, edges=False)` for the vertex version, `intersection_matrix`, `cooccurrence_matrix`, `laplacian(H, kind)` with `kind` one of `"normalized"`, `"randomwalk"`, `"bolla"`, `"clique"`, `adjacency_tensor`, `tensor_apply`, `uniform_layers`, and `simplicial_closure`. The inverse direction is covered by `from_cliques` (maximal cliques of a graph), `from_graph_edges` (a graph as a 2-uniform hypergraph), and `io.from_bipartite`. The readers, writers, and converters of section 2.3 are in `hatvis.io`.

2.5 Examples

What the clique expansion forgets. The script of fig. 2.2 builds the two hypergraphs of fig. 1.1 and compares their representations.

```
H1 = hv.Hypergraph({"group": ["a", "b", "c"]}, name="one 3-way
    interaction")
H2 = hv.Hypergraph({"ab": ["a", "b"], "bc": ["b", "c"], "ac": ["a", "c"]},
    name="three pairwise interactions")
G1, G2 = hv.clique_expansion(H1), hv.clique_expansion(H2)
print("same clique expansion:", set(map(frozenset, G1.edges())) ==
    set(map(frozenset, G2.edges())))
print("same star expansion:  ", nx.is_isomorphic(hv.star_expansion(H1),
    hv.star_expansion(H2)))
print("conformal:", hv.problems.is_conformal(H1),
    hv.problems.is_conformal(H2))
```

The clique expansions are equal, the star expansions are not isomorphic, and the conformality test separates the two: the single group is conformal, the three pairs are not, because the maximal clique $\{a, b, c\}$ of their common expansion is not contained in any of them.

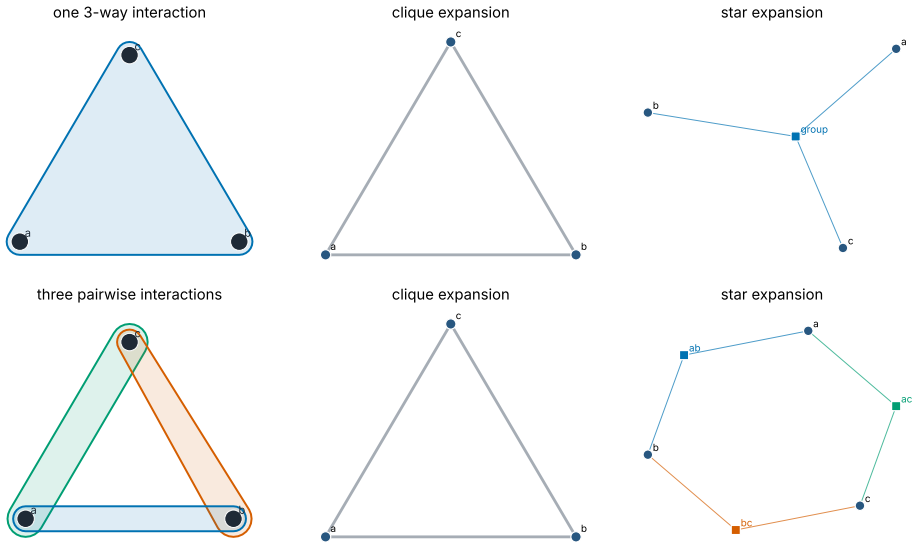


Figure 2.2. What the clique expansion forgets. Top: one interaction of three vertices; bottom: three pairwise interactions. The clique expansions (middle) are identical; the star expansions (right) differ, with one hyperedge node of degree three in the first case and three of degree two in the second.

A production system seen through four graphs. The cyber-physical production example of HyperGodot [2] has fifteen entities (four workers, two robots, five materials, a simulation software, and three machines) and seven tasks, from the 3D design of a component to the final inspection. Figure 2.3 shows its clique expansion, star expansion, and two line graphs. The clique expansion is dense around the components C_2 and C_3, which take part in most tasks, and it hides which entities act together: the five members of the assembly task E_5 form a 5-clique that is indistinguishable from five overlapping triangles. The star expansion

shows the tasks explicitly, as square nodes, and reveals the chain from design (E_1, E_2) through machining and welding (E_3, E_4) to assembly, quality control, and inspection (E_5–E_7). The 1-line graph connects the tasks that share any entity. In the 2-line graph only the tasks that share at least two entities remain linked, and the process falls apart into three parts: the two design tasks E_1 and E_2, which share the component C_1 and the simulation software; the CNC milling task E_3, now isolated; and the four downstream tasks E_4–E_7. The three parts are connected to each other through single components only, which is exactly the kind of weak link that the s-walk analysis of chapter 6 is designed to expose.

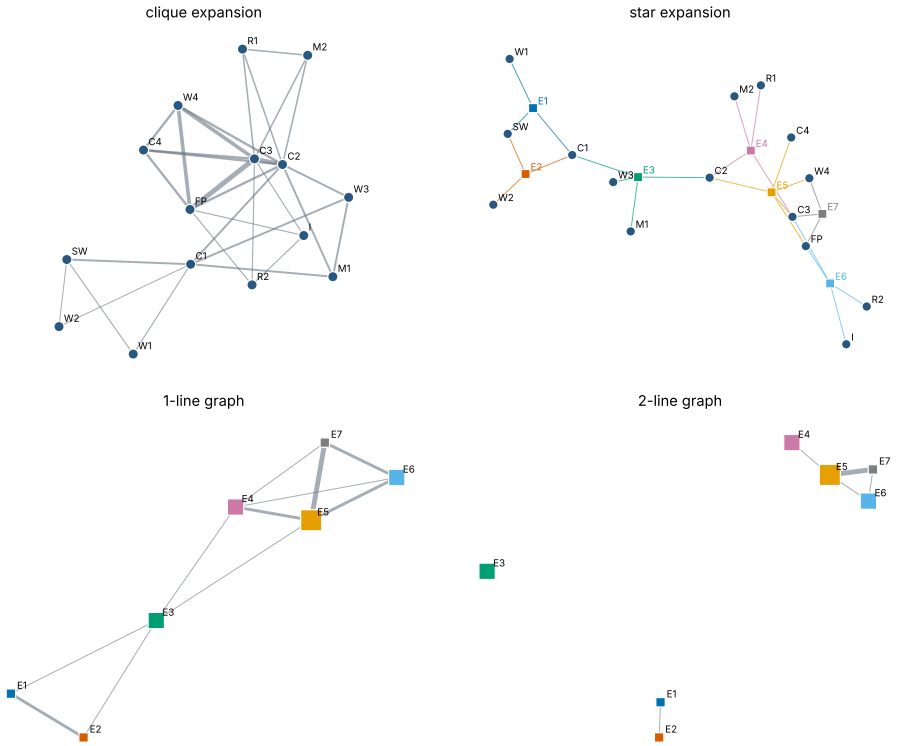


Figure 2.3. Expansions and line graphs of the cyber-physical production example of HyperGodot: clique expansion (top left), star expansion with the seven tasks as squares (top right), 1-line graph (bottom left) and 2-line graph (bottom right) of the tasks. The edge widths of the line graphs are the overlap sizes $|e \cap f|$.

```
spectra = {kind: np.sort(np.linalg.eigvalsh(hv.laplacian(H, kind))) for
    kind in ("normalized", "bolla", "clique")}
for kind, ev in spectra.items():
    print(f"{kind:>10}: zero eigenvalues = {np.sum(ev < 1e-9)}, largest =
    {ev[-1]:.3f}")
print("algebraic connectivity:",
    round(hv.spectral.algebraic_connectivity(H), 4))
```

The three Laplacian spectra of fig. 2.4 each have exactly one zero eigenvalue, so the system is connected. The normalized Laplacian has its largest eigenvalue at 1 and a plateau of eight eigenvalues equal to 1, as predicted by the rank argument of section 2.2.5 ($n - m = 15 - 7 = 8$); its algebraic connectivity is 0.1009. The combinatorial Laplacians have no plateau and much larger spectral radii (73.5 for Bolla's and 314.0 for the clique Laplacian, which counts every shared task of every pair), but after normalization by the largest eigenvalue their shapes almost coincide. For clustering and embedding the normalized operator is usually preferable, because its eigenvalues are comparable across hypergraphs; for diffusion with a physical time scale the combinatorial ones are the natural choice (chapter 11).

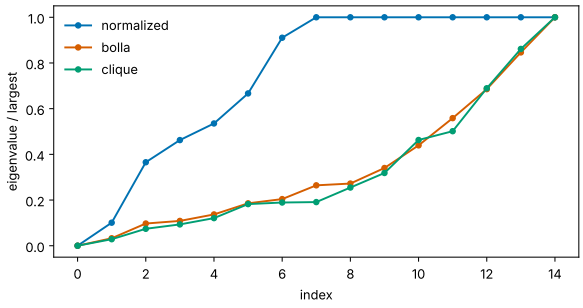


Figure 2.4. Spectra of three Laplacians of the production example, each divided by its largest eigenvalue. The normalized Laplacian of Zhou et al. has eight eigenvalues equal to its maximum 1, because its non-trivial part has rank at most $m = 7$.

Interchange. The last script writes the production example to HIF and to the HyperGodot format, reads both back, and converts it to the three Python packages and back again.

```
io.write_hif(H, tmp / "cps.hif.json")              # shared by HyperNetX,
    XGI, Hypergraphx, HAT
io.write_hypergodot(H, tmp / "cps.txt")             # "W1 C1 SW #WEIGHT:
    15 #GROUP: H"
print((tmp / "cps.txt").read_text().splitlines()[0])
back = io.read_hif(tmp / "cps.hif.json")
print("HIF round trip identical:", back == H)
print("HyperGodot round trip, same hyperedges:", io.read_hypergodot(tmp /
    "cps.txt").edge_sets() == H.edge_sets())
```

The HIF round trip returns an identical object, including the weights (processing times) and the task attributes; the HyperGodot file reproduces the hyperedges, weights, and groups, with the first line `W1 C1 SW #WEIGHT: 15 #GROUP: H`. The conversions to HyperNetX, XGI, and Hypergraphx and back return the same vertex sets. Figure 2.5 shows the degree and size distributions computed after the round trip: nine of the fifteen entities take part in a single task, and the component C_3 in four; three tasks have three members, three have four, and the assembly has five.

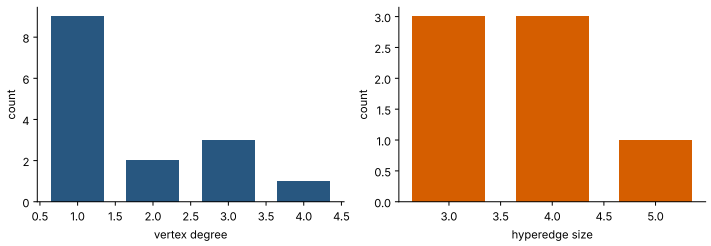


Figure 2.5. Vertex degree (left) and hyperedge size (right) distributions of the production example after a round trip through HIF.

2.6 Further reading

Berge [16] treats conformality, the Helly property, and duality, and Bretto [17] the incidence graph and the 2-section. The s-walk framework and its line graphs are developed by Aksoy et al. [21]. Agarwal et al. [36] compare hypergraph Laplacians with those of the expansions, and Torres et al. [14] discuss the choice between graphs, hypergraphs, and simplicial complexes for a given system. Lee-Kwang and Lee [32] define fuzzy hypergraphs and their α-cuts; Gallo et al. [33] and Ausiello and Laura [34] cover directed hypergraphs. The HIF paper [26] describes the schema and its integration in the major packages.

CHAPTER

3



Matrix views

The incidence matrix is the only drawing of a hypergraph that never hides an incidence and never needs a layout, but in its original order it is a scatter of dots. This chapter treats the reordering of rows and columns (seriation) as an optimization problem, presents the iterative barycentric method of HAT-VIS together with correspondence analysis and hierarchical clustering with optimal leaf ordering, measures the result with the Moore stress and the bond energy, and extracts biclusters, groups of vertices used together by groups of hyperedges. Two set-oriented matrix views close the chapter: the UpSet plot of intersections and the PAOH view of a hypergraph that changes over time. The examples use the wire-harness manufacturing space of [4] and a synthetic collection of news articles.

3.1 Problem and idea

A node-link drawing of a hypergraph with seventy hyperedges is a tangle of overlapping outlines; the incidence matrix of the same hypergraph is a grid of 31×70 cells that can be read row by row, column by column, at any size. Its weakness is that the rows and columns have no natural order. Resources are listed by type, activities by the time they occur, and the blocks that matter (a cell of machines and operators that always work together, a group of activities that use the same tools) are spread over the whole matrix. Permuting rows and columns changes nothing in the data and everything in the picture (fig. 3.1).

The idea of *seriation* is to choose the permutations so that similar rows and similar columns are adjacent, which turns co-occurrence into visible blocks and bands [44, 45]. Once the matrix is ordered, a second step can name the blocks explicitly: a *bicluster* is a set of rows and a set of columns whose submatrix is dense. In a manufacturing space the biclusters are candidate cells; in a document

collection they are topics with their documents; in a market-basket hypergraph they are product families with their customer segments.

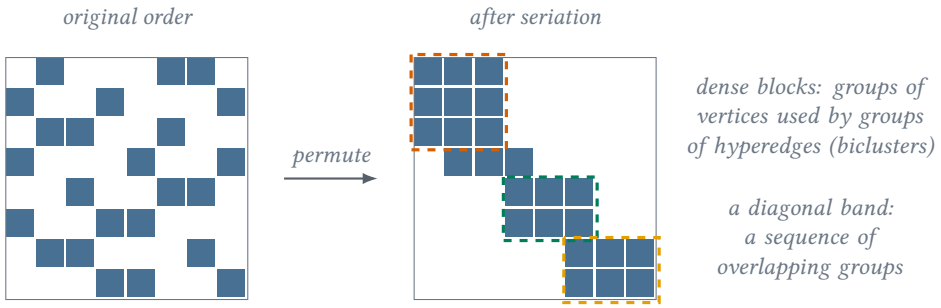


Figure 3.1. Seriation of an incidence matrix. The two matrices contain the same 24 incidences (rows: vertices, columns: hyperedges); only the order differs. After reordering, groups of co-occurring vertices and hyperedges form dense blocks, and a chain of partially overlapping groups forms a band along the diagonal.

3.2 Theory

3.2.1 Seriation as an optimization problem

Let $\mathbf{X} \in \mathbb{R}^{n \times m}$ be the incidence matrix (or its 0/1 pattern) and let π and ρ be permutations of the rows and columns. Seriation chooses (π, ρ) to minimize a loss or maximize a merit of the reordered matrix $\mathbf{X}_{\pi\rho}$. Two measures are standard for binary matrices. The *Moore stress* [46] sums the squared differences between neighboring cells,

$$\mathrm{MS}(\pi, \rho) = \sum_{i=1}^{n-1} \sum_{k=1}^{m} \left(x_{\pi(i)\rho(k)} - x_{\pi(i+1)\rho(k)} \right)^2 + \sum_{i=1}^{n} \sum_{k=1}^{m-1} \left(x_{\pi(i)\rho(k)} - x_{\pi(i)\rho(k+1)} \right)^2, \quad (3.1)$$

and counts the boundaries between filled and empty cells; it is small when the ones form compact blocks. The *bond energy* of McCormick, Schweitzer, and White [47],

$$\mathrm{BE}(\pi, \rho) = \sum_{i,k} x_{\pi(i)\rho(k)} \left(x_{\pi(i+1)\rho(k)} + x_{\pi(i)\rho(k+1)} \right), \quad (3.2)$$

counts the pairs of filled cells that touch and is large for the same arrangements. For a 0/1 matrix the two are related: every incidence has four sides, each either shared with another incidence or a boundary, so maximizing the bonds and minimizing the boundaries pull in the same direction, although the treatment of the matrix border makes them differ in detail. Finding the optimal permutations is NP-hard for both measures (the row problem alone contains the traveling salesman problem), so practical methods are heuristics or relaxations [44, 48]. Three of them are implemented in `hatvis`.

Algorithm 3.1 Barycentric seriation (HAT-VIS)

Input: incidence matrix $\mathbf{X}$; maximum number of iterations (default 10)
Output: row order π and column order ρ

$\mathbf{C} \leftarrow \mathbf{X}\mathbf{X}^{\mathsf{T}}$; $x_i \leftarrow i$;
repeat
  $b_i \leftarrow \sum_j c_{ij} x_j / \sum_j c_{ij}$ for all non-empty rows (3.3);
  $x \leftarrow$ ranks of b (ties broken by the previous rank);
until *the order is unchanged or the iteration limit is reached*;
$\pi \leftarrow$ order of x, empty rows last;
repeat the iteration on $\mathbf{X}^{\mathsf{T}}\mathbf{X}$ to obtain ρ;

3.2.2 Barycentric seriation

The HAT-VIS scripts [1] order the vertices by an iteration on the co-occurrence matrix $\mathbf{C} = \mathbf{X}\mathbf{X}^{\mathsf{T}}$ of (2.1). Starting from the current rank vector $\mathbf{x}$ (initially $x_i = i$), every row receives the weighted average of the ranks of the rows it co-occurs with,

$$b_i = \frac{\sum_j c_{ij}\, x_j}{\sum_j c_{ij}}, \tag{3.3}$$

and the rows are re-ranked by b; the columns are ordered in the same way with $\mathbf{X}^{\mathsf{T}}\mathbf{X}$. The iteration is a discrete form of *reciprocal averaging*: rows that share many hyperedges pull each other's ranks together, and after a few rounds tightly co-occurring rows are adjacent. HAT-VIS runs ten iterations; `hatvis` uses the same default and stops earlier when the order no longer changes (algorithm 3.1). The method is fast, needs only the sparse co-occurrence matrix, and tends to produce a clean order inside the dominant group of rows, at the price of being sensitive to the initial order.

3.2.3 Correspondence analysis

Reciprocal averaging, iterated to convergence with continuous scores instead of ranks, converges to the first non-trivial axis of *correspondence analysis* [49]. Let $\mathbf{P} = \mathbf{X}/\sum_{ik} x_{ik}$, with row sums $\mathbf{r}$ and column sums $\mathbf{c}$, and $\mathbf{D}_r = \operatorname{diag}(\mathbf{r})$, $\mathbf{D}_c = \operatorname{diag}(\mathbf{c})$. The singular value decomposition of the standardized residual matrix

$$\mathbf{D}_r^{-1/2}\left(\mathbf{P} - \mathbf{r}\mathbf{c}^{\mathsf{T}}\right)\mathbf{D}_c^{-1/2} = \mathbf{U}\Sigma\mathbf{V}^{\mathsf{T}} \tag{3.4}$$

gives row scores $\mathbf{D}_r^{-1/2}\mathbf{u}_1$ and column scores $\mathbf{D}_c^{-1/2}\mathbf{v}_1$ from the leading singular vectors; sorting by them yields simultaneous row and column orders. The scores solve a relaxation of the seriation problem in which the ranks are allowed to be real numbers, and they place a row at the weighted average of the columns it

touches and a column at the weighted average of its rows, which is exactly the fixed point of (3.3). Correspondence analysis is the method of choice when the matrix has a single dominant gradient, such as the sequence of activities along a production line; when the matrix consists of several unrelated blocks, a single axis can only order them one after another and may interleave their members.

3.2.4 Hierarchical clustering with optimal leaf ordering

The third method clusters the rows (and separately the columns) hierarchically, here with the Jaccard distance between the 0/1 rows and average linkage, and reads the order off the leaves of the dendrogram. A dendrogram with n leaves admits 2^{n-1} leaf orders, since the two children of every internal node can be swapped; Bar-Joseph, Gifford, and Jaakkola [50] gave a polynomial algorithm that finds the swap pattern minimizing the sum of distances between adjacent leaves. The result keeps every cluster contiguous at every level of the hierarchy, so blocks appear as nested rectangles, and it adapts well to matrices composed of several blocks of different shapes. Its weakness is the opposite of correspondence analysis: a global gradient may be broken at the boundaries of the clusters.

3.2.5 Biclusters

A seriated matrix suggests blocks; *spectral co-clustering* names them. Dhillon [51] treats the incidence matrix as the biadjacency matrix of the star expansion and looks for a partition of its vertices *and* hyperedges into k groups that minimizes the normalized cut of the bipartite graph. The relaxed problem is solved by the singular vectors of the normalized matrix

$$\tilde{\mathbf{X}} = \mathbf{D}_1^{-1/2}\,\mathbf{X}\,\mathbf{D}_2^{-1/2}, \qquad \mathbf{D}_1 = \operatorname{diag}(\mathbf{X}\mathbf{1}),\ \mathbf{D}_2 = \operatorname{diag}(\mathbf{X}^{\mathsf{T}}\mathbf{1}); \tag{3.5}$$

the singular vectors $2, \dots, \ell+1$ with $\ell = \lceil \log_2 k \rceil$, rescaled by $\mathbf{D}_1^{-1/2}$ and $\mathbf{D}_2^{-1/2}$, are stacked for rows and columns, and k-means on the stacked embedding assigns every vertex and every hyperedge to one of k co-clusters. Every co-cluster is a candidate bicluster; `hatvis` reports it with its *density*, the fraction of ones in its submatrix, and drops blocks below a chosen density. Because rows and columns are clustered together, a block of resources comes with the activities that use it, which is the form in which cells of a manufacturing system or topics of a document collection are needed. The same method reappears as a community detection method for vertices and hyperedges in section 7.2.3.

3.2.6 Set-oriented matrix views

Two further matrix layouts are designed for specific questions about sets.

In a nutshell **Matrix views**

- Seriation permutes rows and columns of the incidence matrix to reveal blocks and bands; quality: Moore stress (3.1) (lower is better) and bond energy (3.2) (higher is better). The exact problem is NP-hard.
- Barycentric (HAT-VIS): iterated co-occurrence-weighted averaging of ranks; fast, sparse, depends on the start.
- Correspondence analysis: first non-trivial singular vectors of (3.4); best for one dominant gradient.
- Hierarchical with optimal leaf ordering: nested contiguous clusters; best for several blocks.
- Spectral co-clustering (3.5) assigns vertices and hyperedges jointly to k biclusters; report densities.
- UpSet: columns are combinations, bars their weights. PAOH: rows are vertices, columns hyperedges grouped by time.

UpSet. Lex et al. [52] replaced Venn and Euler diagrams of many sets by a matrix in which every column is a set *intersection*, drawn as a vertical line of dots over the participating sets, with a bar chart of the intersection sizes above and a bar chart of the set sizes on the side. Applied to a hypergraph, the natural reading is that every column is a distinct hyperedge (a combination of vertices) and the bar is its weight or multiplicity; the horizontal bars are the weighted vertex degrees. The view is ideal for frequent itemsets, where the question is which combinations occur and how often (chapter 8), and it scales to dozens of combinations where a Venn diagram fails beyond four sets.

PAOH. The *Parallel Aggregated Ordered Hypergraph* view of Valdivia et al. [53] draws vertices as horizontal lines and every hyperedge as a vertical segment with dots at its members. For a temporal hypergraph the columns are grouped by time stamp and the groups separated by gaps, so the evolution of the groups can be followed without any node-link clutter. The row order is again a seriation problem; `hatvis` uses the barycentric order of the aggregated hypergraph so that vertices that often appear together are adjacent in every period.

3.3 Using it in `hatvis`

```
seriate(H, method="barycentric", weighted=False, n_iter=10)
```

method `"barycentric"` (HAT-VIS), `"correspondence"`, `"hierarchical"` (optimal leaf ordering), or `"none"`.
Returns two label lists: the vertex order and the hyperedge order.
Related `seriation_quality(H, rows, cols)` returns the Moore stress and the bond energy; `bicluster_blocks(H, n_clusters, random_state, min_density)` returns a list of blocks with `nodes`, `edges`, and `density`; the lower-level functions `barycentric_order`, `correspondence_order`, and `hierarchical_order` work on any matrix.

```
plot_incidence(H, order="barycentric", ax=None, transpose=False,
biclusters=None, node_groups=None, labels=True)
```

order a seriation method name, a pair of label lists, or `None` for the stored order.
transpose hyperedges in rows (as in the figures of [4]).
biclusters the output of `bicluster_blocks`, drawn as dashed rectangles.
node_groups a `node_data` column whose values color the vertex labels.
Related `plot_paoh(H, time=None, row_order="barycentric")`, `plot_upset(H, by="edges", max_bars=20)`, `plot_similarity_matrix(S, order)`.

3.4 Examples

The wire-harness manufacturing space. Nagy et al. [4] model an intelligent collaborative manufacturing space as a hypergraph whose vertices are resources and whose hyperedges are activities. Their benchmark is a wire-harness assembly with two parallel production lines; each line takes wires from a Kanban storage through buffers to two crimping and two assembly stations, and an automated guided vehicle (AGV) transports the material between them. The thirty-one resources are seven operators O_1–O_7, six robots R_1–R_6, the AGV, four crimping machines M_1–M_4, two shared tools E_1, E_6, three screwdrivers D_1–D_3, five competences C_1, C_3, C_6, C_7, C_8, and three sensors S_1–S_3; the seventy activities a_1–a_{70} cover loading and unloading the AGV, manual handling, positioning crimps in vises and fixtures, crimping, assembly with robots and screwdrivers, and hand-overs between stations. The original incidence matrix is confidential. `hatvis` ships a reconstruction (`load_wire_harness`) with the same resources, the published activity sequence and types, and resource sets derived from the published process

description; the results below have the same structure as those of the paper but are not its numbers.

Seriation. The script of fig. 3.2 orders the matrix with the three methods and evaluates each order.

```
quality = {}
orders = {"original": (H.nodes, H.edges)}
for method in ("barycentric", "correspondence", "hierarchical"):
    orders[method] = hv.seriate(H, method)
for name, (rows, cols) in orders.items():
    quality[name] = hv.seriation.seriation_quality(H, rows, cols)
print(pd.DataFrame(quality).T)
```

Table 3.1. Seriation quality of the wire-harness incidence matrix (31 resources, 70 activities). Lower Moore stress and higher bond energy are better.

Order	Moore stress	Bond energy
original	644	162
barycentric (HAT-VIS)	569	201
correspondence analysis	579	196
hierarchical, optimal leaf order	398	289

Table 3.1 and fig. 3.2 show the outcome. All three methods improve on the original order, in which the activities are listed by time and the resources by type, so that the operators of the two lines and the stations they serve are far apart. The barycentric and correspondence orders give similar values: both lay out a dominant gradient as a band and gather the transport activities, which share the AGV, at one end. Hierarchical clustering with optimal leaf ordering reduces the Moore stress by 38% and raises the bond energy by 78% relative to the original order. The reason is visible in the figure: the matrix consists of several nearly separate blocks (transport, crimping, assembly, and the per-line stations) of different shapes, which is the situation in which a nested, cluster-preserving order beats a single global axis.

Biclusters. Spectral co-clustering with six co-clusters, keeping the blocks of density at least 0.3, finds five biclusters.

```
blocks = hv.seriation.bicluster_blocks(H, n_clusters=6, random_state=0,
    min_density=0.3)
for b in blocks:
    print(f"{len(b['nodes']):2d} resources x {len(b['edges']):2d}
    activities, density {b['density']:.2f}:",
          ", ".join(b["nodes"]))
order = hv.seriate(H, "correspondence")
```

The densest block (0.86) joins the robot R_2 and the crimping machine M_2 with the seven activities of the second crimping station of line 1, from taking the wire

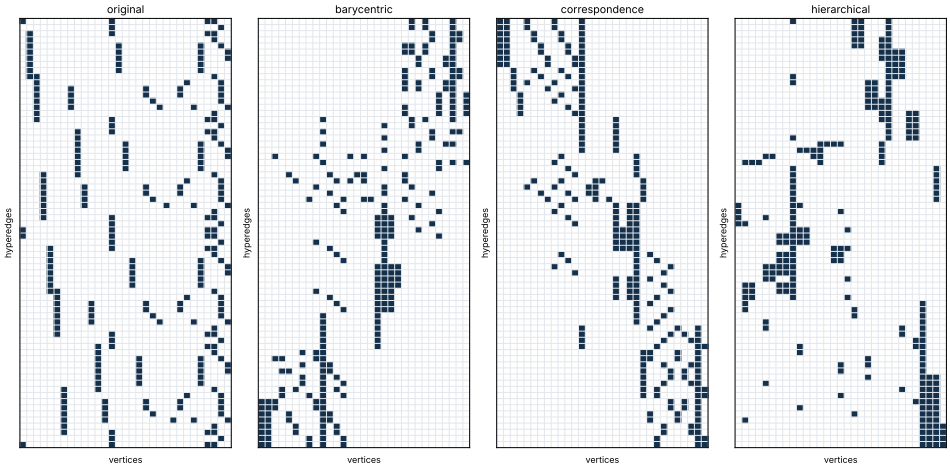


Figure 3.2. Seriation of the wire-harness incidence matrix, activities in rows and resources in columns. From left to right: original order, barycentric seriation of HAT-VIS, correspondence analysis, and hierarchical clustering with optimal leaf ordering. The quality measures are listed in table 3.1.

out of the buffer to handing the crimped wire over to assembly; the second (density 0.83) is the transport cell, the operator O_1, the AGV, the loading competence C_8, and the sensor S_1, with the twelve loading and unloading activities of the six AGV transports. A third, smaller block joins O_4, R_3, and the screwdriver D_2 with the five activities of the second assembly station of line 1. The two remaining blocks are larger and sparser: a crimping group of eight resources (operators, robots, and machines of the other crimping stations with the crimping competence C_7 and the machine-log sensor S_3) and an assembly group of fourteen resources. Figure 3.3 outlines the blocks on the correspondence-ordered matrix. The structure matches the reading of the paper: the AGV transport, the crimping stations, and the assembly stations are the modules of the system, and the less dense blocks are the places where resources are shared between stations or lines, the natural candidates for reorganization.

A dynamic hypergraph without clutter. The collection of news articles used in chapters 8 and 12 is synthetic, a stand-in for event data of the GDELT type [54]. Each article is a hyperedge whose vertices are its themes; there are twenty-four themes in six planted narratives (climate science, energy, finance, policy, impacts, and society, four themes each). An article mixes one or, with probability 0.4, two narratives, takes between two and four themes from each, and with probability 0.35 adds a random background theme. The popularity of the narratives changes over three periods (2021–2023): finance and loss-and-damage grow strongly, impacts and society grow more slowly, and energy declines. Figure 3.4 shows fourteen articles per period in the PAOH view.

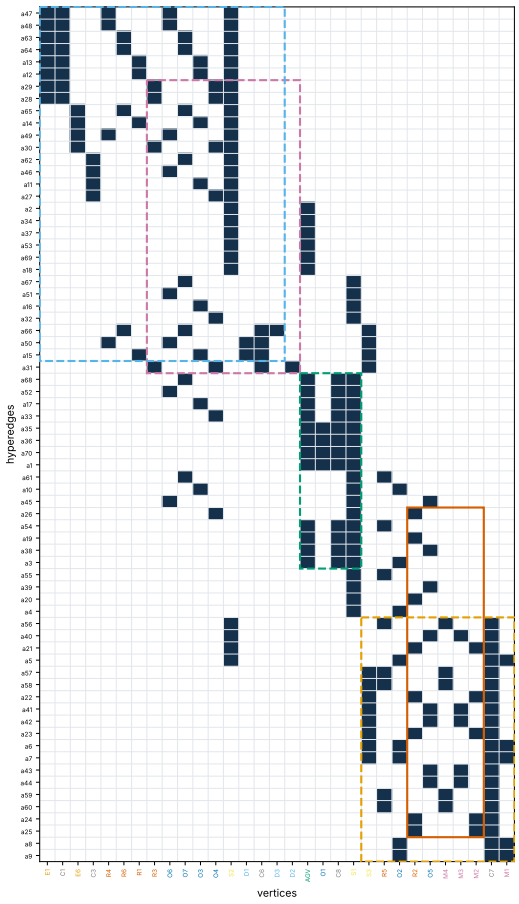


Figure 3.3. Biclusters of the wire-harness space found by spectral co-clustering, outlined on the incidence matrix ordered by correspondence analysis (activities in rows). The resource labels are colored by type.

```
ax = plot_paoh(T, time="time")
```

The barycentric row order places the themes of every narrative next to each other, so each article appears as one or two short vertical runs, and the periods can be compared at a glance: the climate-science, energy, and policy combinations dominate the first period, while articles that combine finance, aid, loss and damage, and developing countries become frequent in the last.

Frequent combinations. With 150 articles per period, the frequent theme combinations of size at least two are mined (chapter 8) and the eighteen most frequent are shown as an UpSet plot (fig. 3.5).

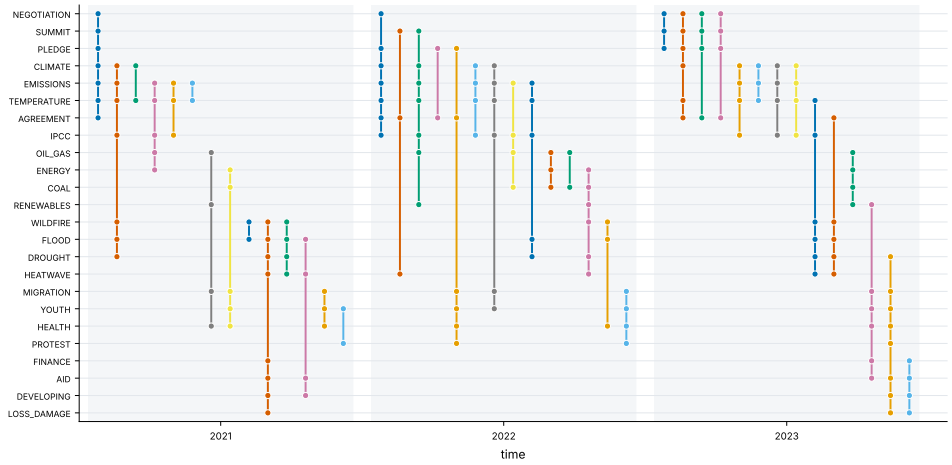


Figure 3.4. PAOH view of 42 synthetic news articles over three periods. Rows are the 24 themes in barycentric order, columns the articles grouped by year; each vertical segment is one article with dots at its themes.

```
fi = hv.frequent_itemsets(T, min_support=0.06, min_size=2)
IH = hv.itemset_hypergraph(fi.head(18))
fig, axes = plot_upset(IH, title="the 18 most frequent theme combinations")
```

The most frequent combinations are pairs from the climate-science narrative ({EMISSIONS, TEMPERATURE} with support 0.18, {EMISSIONS, CLIMATE} with 0.17), followed by pairs from energy ({OIL_GAS, COAL}, 0.16) and impacts ({DROUGHT, FLOOD}, 0.16). The dot matrix shows at once that almost all frequent combinations stay inside one narrative, the signature of the planted structure, whereas the vertex bars on the left show which themes take part in many combinations. Exactly this table is hard to read from a node-link drawing, where eighteen overlapping hyperedges on the same few vertices are indistinguishable.

3.5 Further reading

Liiv [44] traces the history of seriation from archaeology to bioinformatics, and Behrisch et al. [45] survey and compare matrix reordering algorithms for visualization; the R package *seriation* [48] implements many of them with a common interface and quality measures. Hill [49] explains the connection between reciprocal averaging and correspondence analysis. Spectral co-clustering is due to Dhillon [51]. UpSet [52] and PAOH [53] are described with extensive user studies in their original papers, and Alsallakh et al. [30] place them among the set visualization techniques.

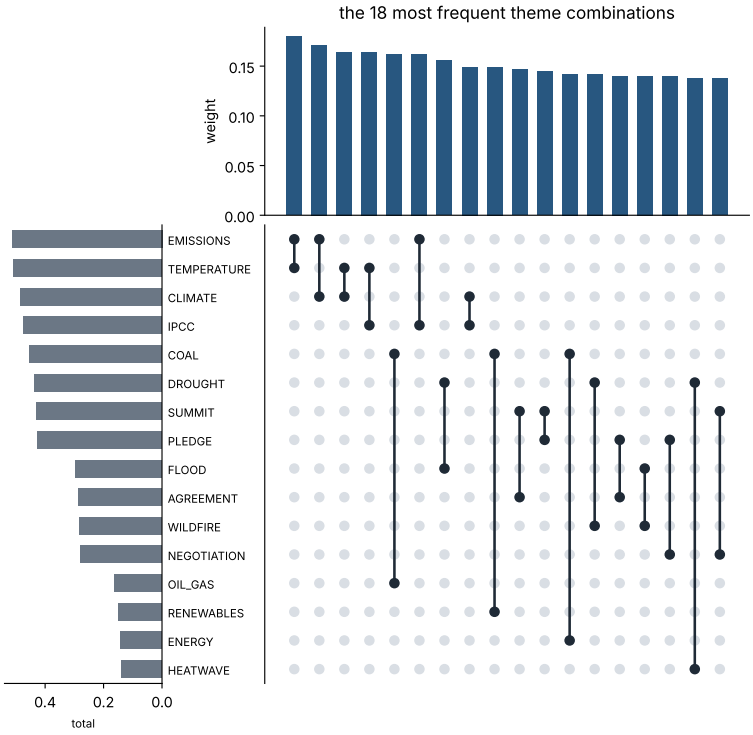


Figure 3.5. UpSet view of the eighteen most frequent theme combinations in 450 synthetic news articles. Each column is a frequent itemset with its support as the bar height; the horizontal bars give the total support of the combinations each theme takes part in.

CHAPTER

4



Node-link and set views

A node-link drawing of a hypergraph has two independent parts: the positions of the vertices (the layout) and the shapes that show the hyperedges around them. This chapter treats them separately. It presents the force-directed and circular layouts of HyperGodot together with the classical spectral, radial, and bipartite layouts, defines two measures that make layouts comparable for hypergraph drawing (stress and intrusions), and then discusses seven ways to draw a hyperedge, from the padded convex hull to implicit isocontours, ellipses, stars, and minimum spanning trees. Comparative views by groups and line styles, linked primal and dual views, a circular hub drawing, and an interactive HTML viewer complete the toolkit. The similarity-based layouts of HAT-VIS and the membership-aware layout are the subject of chapter 5.

4.1 Problem and idea

Drawing a graph means placing vertices and drawing line segments between them; the segments are unambiguous because each connects exactly two points. A hyperedge connects an arbitrary number of points, and every way of drawing it is a compromise. A closed curve around the members (the *subset standard* of Euler-type diagrams [55, 56]) is the most direct encoding, but it can enclose vertices that do not belong to the hyperedge, and with many overlapping hyperedges the curves become hard to follow. Johnson and Pollak [57] showed that deciding whether a hypergraph admits a planar drawing of this kind is NP-complete, so perfect drawings cannot be expected in general. An extra node per hyperedge with spokes to the members (the drawing of the star expansion) avoids false enclosures but turns a hypergraph into a graph again and doubles the number of nodes. Trees or polygons through the members are light but suggest an internal structure that the hyperedge does not have.

The idea of this chapter, inherited from HyperGodot [2], is to decouple the two decisions. The layout is computed from the hypergraph alone and evaluated by explicit measures; the hyperedge style is chosen afterwards for the question at hand and can be changed without moving a single vertex (fig. 4.1). Attributes enter through visual variables that do not interfere with the layout: line style for a group, line width for a weight, vertex size for a degree or a centrality, color for a category.

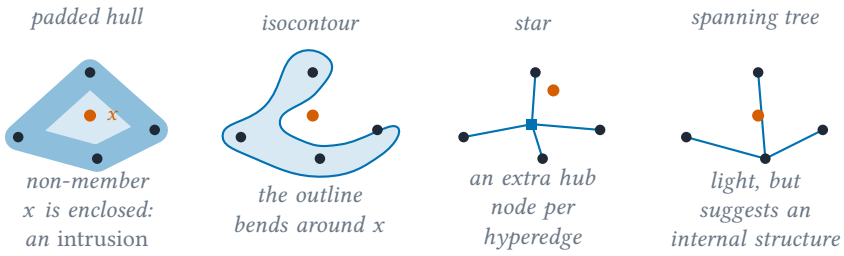


Figure 4.1. Four ways to draw the same four-vertex hyperedge on the same layout. The padded convex hull (HyperGodot) encloses the non-member x; an implicit isocontour (Bubble Sets) bends around it; the star with a hub node never encloses anything but adds a node; the minimum spanning tree of the members (HAT-VIS) is the lightest mark but looks like a path structure inside the group.

4.2 Theory

4.2.1 Force-directed layout

Force-directed layouts treat the drawing as a physical system whose equilibrium is a good picture [58]. For hypergraphs the attraction has to act on groups, not on pairs; one option is to run a graph layout on the star expansion, where every hyperedge node pulls its members together [59], another is to attract every vertex to the centroid of each of its hyperedges. HyperGodot [2] uses the second. With $\mathbf{p}_i$ the position of vertex i and $\mathbf{c}_e = \frac{1}{|e|}\sum_{j\in e}\mathbf{p}_j$ the centroid of hyperedge e, the force on vertex i is

$$\mathbf{F}_i = \sum_{e\ni i} k_s\big(\,\|\mathbf{c}_e-\mathbf{p}_i\|-l_0\big)\,\frac{\mathbf{c}_e-\mathbf{p}_i}{\|\mathbf{c}_e-\mathbf{p}_i\|} \;+\; \sum_{j\neq i}\frac{k_r}{\|\mathbf{p}_i-\mathbf{p}_j\|^2}\,\frac{\mathbf{p}_i-\mathbf{p}_j}{\|\mathbf{p}_i-\mathbf{p}_j\|}, \tag{4.1}$$

a Hooke spring of constant k_s and rest length l_0 toward every centroid and an inverse-square repulsion of strength k_r from every other vertex. Each iteration moves every vertex by the damped force, $\mathbf{p}_i \leftarrow \mathbf{p}_i + \gamma\mathbf{F}_i$, and `hatvis` caps a single displacement for numerical safety. The defaults are those of HyperGodot (500 iterations, $k_r = 250$, $l_0 = 30$ in pixel units, $k_s = 0.1$, $\gamma = 0.9$). The paper states the repulsion with an inverse-distance magnitude, while its source code uses the inverse square; `hatvis` follows the code, which yields the published pictures.

Because the spring acts on the centroid, a vertex of many hyperedges is pulled toward the middle of all of them and ends up at their junction, which is where a reader expects a shared resource; vertices of a single hyperedge orbit its centroid at roughly the rest length.

4.2.2 Other layouts

Four classical layouts complete the set. The *circular* layout places the vertices evenly on a circle. In arbitrary order the hyperedges then cut across the disc; in the barycentric order of section 3.2.2 the members of a hyperedge tend to be neighbors on the circle and the hyperedges stay short. The *radial* layout of HyperGodot places the vertex of largest degree at the center and the others on rings by their breadth-first distance in the clique expansion, which shows how far each entity is from the most involved one. The *spectral* layout uses the two leading non-trivial eigenvectors of the normalized Laplacian (2.4) [35] as coordinates; it is deterministic and reveals the coarse partition, but vertices of the same hyperedges collapse onto each other. The *bipartite* layout runs a spring embedder on the star expansion and draws only the vertices, using the hyperedge nodes as hidden attractors.

4.2.3 Measuring a layout

Aesthetic criteria for graph drawings (few crossings, uniform edge lengths, symmetry) do not transfer directly to hypergraphs. Two properties matter most for the drawings in this book and can be measured.

The first is *faithfulness*: vertices that are similar in the hypergraph should be close in the plane. With the dissimilarities δ_{ij} of chapter 5 (by default one minus the transitive weighted Jaccard similarity) and the Euclidean distances d_{ij} of the layout, the normalized *stress* after optimal scaling is

$$\text{stress} = \min_{\alpha>0} \frac{\sum_{i<j}(d_{ij} - \alpha\,\delta_{ij})^2}{\sum_{i<j} d_{ij}^2}, \tag{4.2}$$

a scale-free version of Kruskal's stress [60]; it is zero when the layout reproduces the dissimilarities up to scale.

The second is *legibility of the hyperedges*: a hull should contain its members and nothing else. An *intrusion* is a pair (j, e) with $j \notin e$ such that vertex j lies inside the padded convex hull of e. The number of intrusions, the number of hyperedges with at least one intrusion, and the mean hull area summarize how readable the hull drawing of a layout will be. A layout with zero intrusions can be drawn with convex hulls without any false enclosure; a layout with many intrusions needs non-convex outlines (isocontours) or a different hyperedge style. The two measures pull in different directions, since a faithful layout may place a non-member between two members it resembles, and section 5.2.5 presents a layout that trades a little stress for fewer intrusions.

4.2.4 Drawing a hyperedge

`hatvis` offers seven hyperedge styles on any layout.

Padded convex hull. HyperGodot draws a hyperedge as the convex hull of discs around its members, computed with Andrew's monotone-chain algorithm [61]. Where several hyperedges share a vertex, their hulls would coincide at that vertex; HyperGodot therefore gives each successive hyperedge a slightly larger disc at a shared vertex, so that the outlines form concentric rings and every hyperedge remains traceable through the junction. In `hatvis` the hyperedges are drawn smallest first, and the disc of vertex i in hyperedge e has radius

$$r_{ie} = p\,\big(1 + \eta\,\ell_i(e)\big), \tag{4.3}$$

where p is the base padding (3% of the layout extent by default), η the padding step (0.45), and $\ell_i(e)$ the number of hyperedges containing i that were drawn before e. The *smooth* variant rounds the hull corners by Chaikin's corner cutting [62].

Implicit isocontour. Bubble Sets [63] draw a set as an isocontour of a potential field that is positive near the members and negative near the non-members, so that the outline bends around intruders. `hatvis` uses the field

$$f_e(\mathbf{x}) = \sum_{j \in e} k_\sigma(\mathbf{x} - \mathbf{y}_j) \;-\; \eta \sum_{j \notin e} k_\sigma(\mathbf{x} - \mathbf{y}_j), \qquad k_\sigma(\mathbf{u}) = \exp\!\big(-\|\mathbf{u}\|^2 / 2\sigma^2\big), \tag{4.4}$$

thresholded at half the kernel peak, and keeps the contour pieces that contain a member. The outline may split into several pieces when the members are far apart, which is itself informative: the layout does not support the hyperedge.

Minimum-volume ellipse. The HAT-VIS scripts outline a hyperedge by the smallest ellipse that contains its members. The minimum-volume enclosing ellipse is computed by Khachiyan's algorithm [64, 65], slightly inflated, and for two members replaced by an elongated ellipse around the segment. Ellipses are smooth and easy to follow but enclose more area than hulls and therefore more intruders.

Polygon, star, and tree. A filled convex polygon of the members (the style of XGI and of the polygon layouts of Qu et al. [66]) is the most compact mark. The star style places a hub at the centroid and draws spokes to the members, the drawing of the star expansion. The tree style draws the Euclidean minimum spanning tree of the members [67], as HAT-VIS does; it connects the members with the least ink. Each of these three can be combined with line widths that encode weights.

4.2.5 Comparative, linked, and interactive views

Groups and line styles. HyperGodot introduced group tags to overlay several hypergraphs, or several versions of one system, in the same drawing: every hyperedge carries a tag, and the tag selects the line style (solid, dashed, dotted). In

In a nutshell **Node-link drawing**

- Layout and hyperedge style are independent; choose the layout by its measures and the style by the question.
- Force-directed (HyperGodot): springs toward hyperedge centroids, inverse-square repulsion (4.1); shared vertices end up at junctions.
- Measures: stress against the similarity dissimilarities (4.2) and intrusions (non-members inside a padded hull).
- Styles: padded hull with concentric padding (4.3), smooth hull, isocontour (4.4), minimum-volume ellipse, polygon, star, spanning tree.
- Attributes: line style for groups, line width for weights, vertex size for degrees or centralities; vertex filter for one entity.
- Linked primal–dual views, circular hub drawing, interactive HTML viewer.

`hatvis` any hyperedge attribute can serve as the group, and line widths (weights), vertex sizes (degrees or centralities), and colors remain free for other variables. The vertex filter of HyperGodot, called the ego view in HAT-VIS, draws the hyperedges of one chosen vertex in color and all others in light gray.

Linked primal and dual views. A vertex of H is a hyperedge of H^* and vice versa. Qu et al. [66] showed that drawing the two side by side, with the color of a hyperedge in one view equal to the color of the corresponding vertex in the other, helps users switch between the entity-centered and the relation-centered reading. `draw_primal_dual` computes both layouts and links the colors.

Circular hub drawing. For hypergraphs with up to a few dozen vertices, a circular drawing in which every hyperedge is a hub inside the circle with curved spokes to its members on the circle gives a compact overview that never encloses a vertex. With the seriated vertex order the spokes are short.

Interactive viewer. `to_html` writes a single self-contained HTML file with a vector drawing of the hypergraph: vertices can be dragged and the hulls follow, the view zooms and pans, a vertex filter highlights the hyperedges of one entity, group tags can be switched on and off, the precomputed layouts can be exchanged, the padding and the smoothness of the hulls can be adjusted with sliders, and the current view can be exported as an SVG file. The file needs no server and no installation and can be sent to colleagues who do not use Python; it is the browser successor of the HyperGodot interface.

4.3 Using it in `hatvis`

```
draw_hypergraph(H, pos=None, *, layout="similarity",
style="hull", ax=None, node_color=None, size_by="degree",
edge_color=None, width_by="weight", group=None, pad=0.03,
pad_step=0.45, labels=True, edge_labels=False, edges=None,
highlight=None)
```

pos, layout a position dictionary, or the name of a layout: `"similarity"`, `"force"`, `"membership"`, `"circular"`, `"spectral"`, `"radial"`, `"bipartite"`, `"grid"`.

style `"hull"`, `"smooth"`, `"bubble"`, `"ellipse"`, `"polygon"`, `"star"`, `"mst"`.

group an `edge_data` column whose values select line styles.

size_by, width_by `"degree"`, `"weight"`, a `Series`, or a column name.

highlight a vertex whose hyperedges are drawn in color and all others in gray.

Related `layout_force`, `layout_circular`, `layout_radial`, `layout_spectral`, `layout_bipartite`, `layout_quality(H, pos)`, `draw_primal_dual`, `draw_circular`, `draw_bipartite`, `draw_clique`, `draw_line_graph`, `to_html(H, path, layouts, group)`.

4.4 Examples

Six layouts. The production example of section 2.5 is laid out in six ways and each layout is evaluated.

```
layouts = {name: resolve_layout(H, None, name, random_state=0) for name in
    names}
quality = pd.DataFrame({name: hv.layout_quality(H, pos) for name, pos in
    layouts.items()}).T
print(quality.round(3))
```

Table 4.1 and fig. 4.2 show that no classical layout wins on both measures. The similarity layout of HAT-VIS is faithful (stress 0.080) but places the robot R_2 and the inspection system I on the same point and produces two intrusions. The force-directed layout of HyperGodot spreads the vertices evenly, with larger hulls and one intrusion. The spectral layout is compact but collapses vertices of the same tasks and has six intrusions in five hyperedges. The circular and bipartite layouts have no intrusions, the first because convex hulls of points on a circle never contain another point of the circle (they may contain parts of other hulls), the second because the hidden hyperedge nodes keep the members together, but both are less faithful. The membership-aware layout of section 5.2.5 has both the lowest stress and no intrusions on this example.

Table 4.1. Quality of six layouts of the production example (15 entities, 7 tasks). Stress is measured against the transitive weighted Jaccard dissimilarities; an intrusion is a non-member inside a padded hull.

Layout	Stress	Intrusions	Hyperedges with intrusions	Mean hull area
similarity (HAT-VIS)	0.080	2	2	0.034
force (HyperGodot)	0.126	1	1	0.091
membership-aware	0.051	0	0	0.049
circular	0.084	0	0	0.089
spectral	0.100	6	5	0.037
bipartite	0.137	0	0	0.026

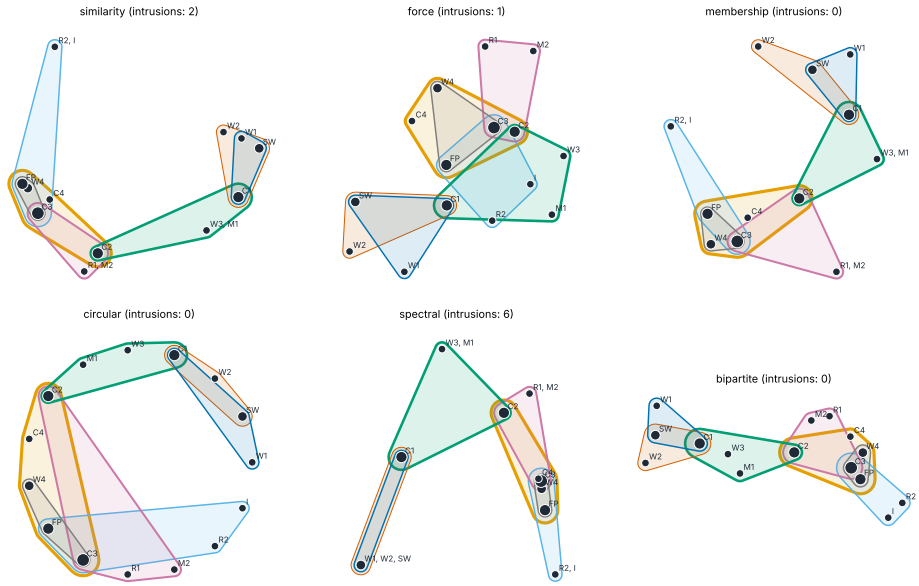


Figure 4.2. Six layouts of the production example drawn with padded convex hulls. The number of intrusions (non-members inside a hull) is given in each title; table 4.1 lists all measures.

Seven styles. With the vertex positions fixed (the membership-aware layout), each hyperedge style is a single argument.

```
draw_hypergraph(H, pos=pos, style=style, ax=ax, legend=False,
title=style, labels=False)
```

Figure 4.3 compares the styles. The padded and smooth hulls show the concentric rings at the shared components C_2 and C_3, so that the four tasks meeting at C_3 can be told apart. The isocontours are tighter than the hulls and separate the long, thin hyperedges better; the ellipses are easy to follow but overlap more.

Polygons and trees use the least ink and are suitable when the hyperedges are small and numerous; the star style makes the tasks visible as nodes, which helps when the tasks themselves carry attributes. The last panel applies the vertex filter to the component C_3: its four tasks (assembly, robotic welding, quality control, inspection) are drawn in color, the rest in gray.

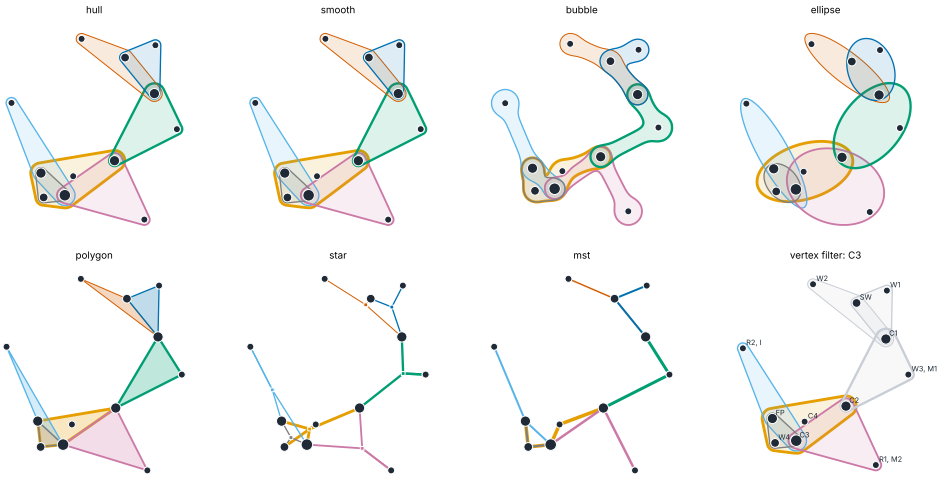


Figure 4.3. The production example drawn with seven hyperedge styles on the same membership-aware layout, and with the vertex filter of HyperGodot applied to the component C_3 (bottom right).

Human and machine work. In the production example every task is tagged as human (H) or machine (M) work, following the publication, and carries a processing time. The script summarizes the two groups and draws them with different line styles, line widths proportional to the processing times, and vertex sizes proportional to the number of tasks.

```
print(hv.temporal.group_statistics(H, "group", weight_name="minutes"))
fig, ax = plt.subplots(figsize=(8, 6.5))
draw_hypergraph(H, layout="force", group="group", width_by="weight",
    size_by="degree", edge_labels=True,
                random_state=3, ax=ax)
```

The human group has five tasks with a total of 85 minutes and the machine group two tasks with 45 minutes, which are the totals reported in [2]; only the assembly time (30 minutes) is given individually in the paper, and the other individual times in the data set are illustrative values that respect the totals. The component C_3 is involved in the largest number of tasks (four). In fig. 4.4 the dashed machine tasks (CNC milling and robotic welding) sit between the design tasks and the assembly-and-inspection cluster, so the drawing shows that the automated steps are the bridge between the two human-dominated phases of the

process. Section 12.4 compares this process with a redesigned version in the same way.

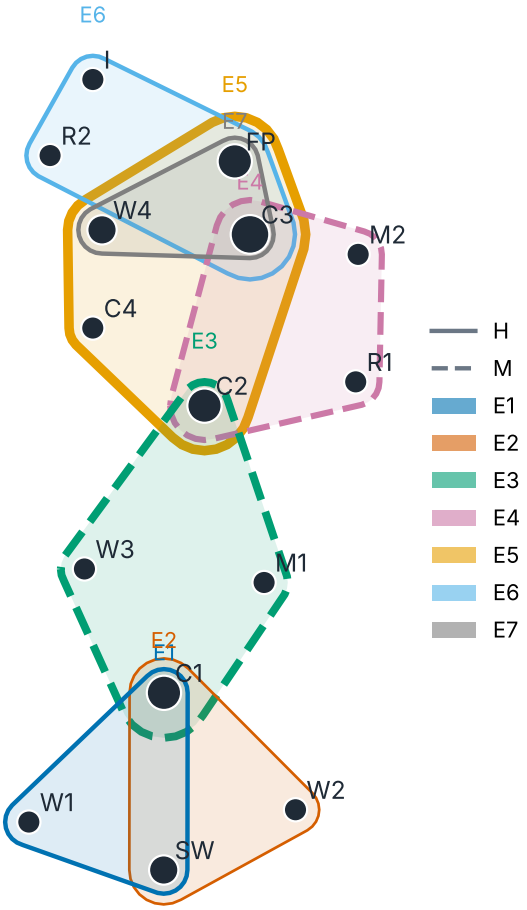


Figure 4.4. Human (solid) and machine (dashed) tasks of the production example on the force-directed layout of HyperGodot. Line widths are processing times, vertex sizes the number of tasks of each entity.

Primal and dual. Figure 4.5 shows the production example and its dual side by side with linked colors.

```
axes = draw_primal_dual(H, layout="similarity", style="smooth",
    labels=True, pad=0.025)
```

In the dual view the seven tasks are the vertices and every entity is a hyperedge around the tasks it takes part in. Entities that take part in a single task become singleton hyperedges (small rings around one task), and the components C_2 and C_3,

which connect several tasks, become the large hyperedges that hold the process together. The dual makes the question "which tasks would be affected if this entity failed" immediate.

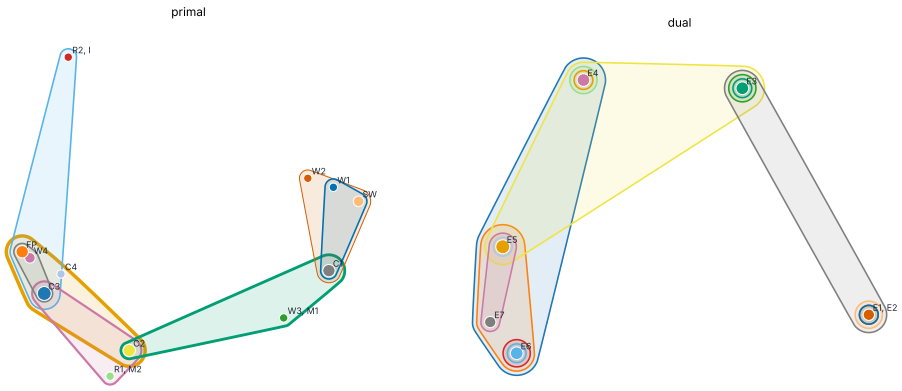


Figure 4.5. Linked primal (left) and dual (right) views of the production example. In the dual the tasks are vertices and every entity is the hyperedge of its tasks; the color of a task hyperedge on the left is the color of the task vertex on the right.

Circular drawing and the interactive viewer. The circular hub drawing of fig. 4.6 shows the effect of the vertex order: with the seriated order, the members of every task are neighbors on the circle and the hubs sit close to the rim; in the original order (workers, robots, components, software, machines) every task spans the whole circle. The same script writes the interactive viewer.

```
path = pathlib.Path(tempfile.mkdtemp()) / "production.html"
hv.to_html(H, path, group="group", title="Cyber-physical production
    (HyperGodot example)")
```

The resulting file of about 13 kB contains the data, four precomputed layouts, and the viewer code; it opens in any browser.

4.5 Further reading

Mäkinen [55] and Bertault and Eades [56] define the subset and edge standards for drawing hypergraphs, and Johnson and Pollak [57] prove the hardness of planar Venn-type drawings. Rodgers [68] surveys Euler diagrams, Alsallakh et al. [30] set visualization in general, and Fischer et al. [31] the hypergraph-specific literature. Qu et al. [66] optimize polygon layouts jointly for the primal and the dual; Oliver et al. extend them to large hypergraphs by simplification [69] and preserve cycles, bridges, and branches during simplification [70]; Zhou et al. [71] simplify hypergraphs with tools of topological data analysis before drawing them. Bubble Sets are described by Collins et al. [63], and force-directed drawing of hypergraphs through the star expansion by Arafat and Bressan [59].

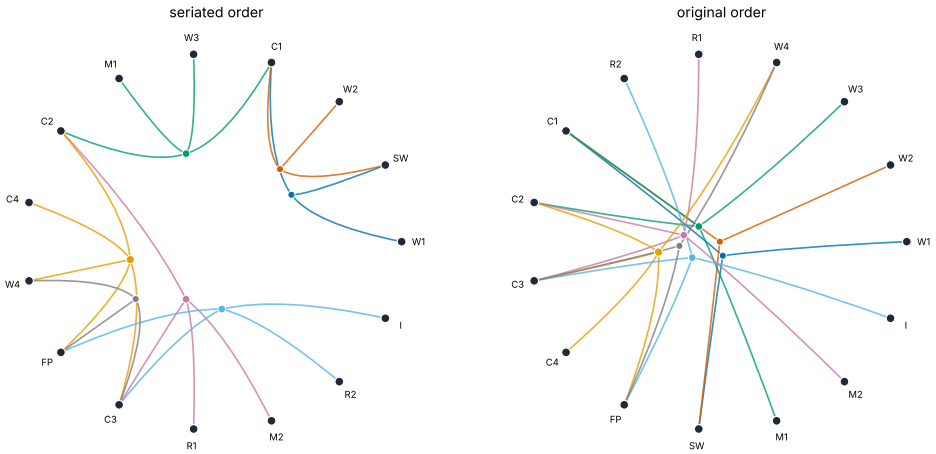


Figure 4.6. Circular hub drawings of the production example with the seriated vertex order (left) and the original order (right). Each task is a hub with curved spokes to its members.

CHAPTER

5



Similarity-based visualization

HAT-VIS draws a hypergraph by placing vertices according to how similar their memberships are, so that the distance in the picture carries meaning. This chapter develops the pipeline step by step: the weighted Jaccard similarity of incidence profiles, its max-product transitive closure, the embedding of the dissimilarities by classical multidimensional scaling or Sammon mapping, and the drawing of hyperedges as minimum spanning trees. It then introduces the membership-aware layout of `hatvis`, which keeps the similarity geometry while pushing non-members out of the hyperedges, and applies the pipeline to the house of quality of an autonomous electric car, the case study of the HAT-VIS paper.

5.1 Problem and idea

A force-directed layout places vertices so that the drawing looks balanced; the distance between two vertices has no direct meaning. For many questions, however, the distance is exactly what one wants to read: two customer requirements that are served by the same engineering solutions should appear close, two resources that are always used together should overlap, and a resource that links two otherwise separate groups should sit between them. HAT-VIS [1] achieves this by deriving a similarity between vertices from their incidence profiles and placing the vertices so that their planar distances approximate the dissimilarities (fig. 5.1). The layout is then a map of the similarity structure, and the hyperedges are drawn on top of it.

The approach has two consequences that this chapter addresses. Vertices with identical memberships become identical points (section 2.2.4), which is correct but needs care in the drawing. And a layout that is faithful to similarities can still place a non-member inside the outline of a hyperedge, because the non-member may

resemble the members without belonging to the group; the membership-aware layout of section 5.2.5 reduces such intrusions without giving up the meaning of the distances.

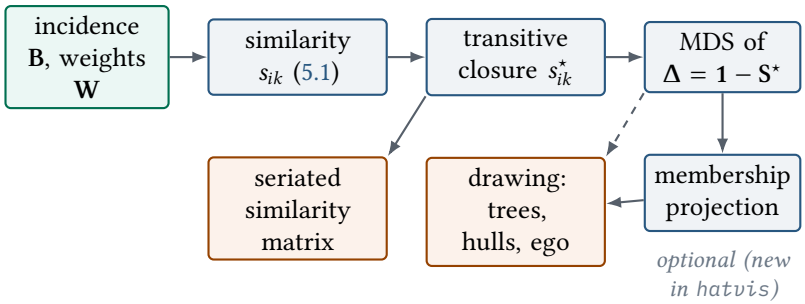


Figure 5.1. The similarity-based visualization pipeline of HAT-VIS with the membership-aware extension of `hatvis`. The weighted Jaccard similarity of the incidence rows is closed transitively, turned into a dissimilarity, and embedded in the plane; the hyperedges are drawn on the resulting positions. The optional projection step moves non-members out of the hyperedges while keeping the similarity geometry.

5.2 Theory

5.2.1 Similarity of incidence profiles

A vertex v_i is described by its row of the incidence matrix, scaled by the hyperedge weights: the *profile* $\mathbf{a}_i = (w_e b_{ie})_{e \in E}$. For a plain hypergraph the profile is the indicator vector of the hyperedges that contain v_i; for the house of quality below it is the vector of weighted relationship scores. HAT-VIS compares profiles with the *weighted Jaccard* (Ruzicka) similarity

$$s_{ik} = \frac{\sum_e \min(a_{ie}, a_{ke})}{\sum_e \max(a_{ie}, a_{ke})}, \tag{5.1}$$

which for 0/1 profiles reduces to Jaccard's coefficient [72] $|E_i \cap E_k| / |E_i \cup E_k|$, the number of shared hyperedges divided by the number of hyperedges of either vertex. The similarity is 1 for identical profiles and 0 for vertices without a common hyperedge. `hatvis` offers the alternatives that the literature uses most often: the Dice coefficient $2 \sum \min / (\sum a_{ie} + \sum a_{ke})$, the overlap coefficient (normalized by the smaller profile, so that a vertex whose hyperedges are a subset of another's has similarity 1 with it), the cosine of the profiles, the asymmetric inclusion $\sum \min / \sum a_{ie}$, the Hamming similarity of the 0/1 patterns, and the plain count of shared hyperedges. Applied to the dual hypergraph, the same measures compare hyperedges by their members.

5.2.2 Transitive closure

Jaccard similarity is local: two vertices that never share a hyperedge have similarity zero, however strongly they are linked through others. In the small example of section 1.6, v_1 and v_5 share no hyperedge, so $s_{15} = 0$, although v_1 is similar to v_3 ($s_{13} = 1/3$) and v_3 to v_5 ($s_{35} = 1/2$). HAT-VIS replaces S by its *max-product transitive closure*

$$s^{\star}_{ik} = \max_{P\colon i \to k} \prod_{(u,v)\in P} s_{uv}, \tag{5.2}$$

the similarity of the strongest chain of pairwise similarities from v_i to v_k. In the example $s^{\star}_{15} = s_{13}\, s_{35} = 1/6 \approx 0.167$. Taking logarithms turns (5.2) into a shortest-path problem with edge lengths $-\log s_{uv} \geq 0$, so $\mathbf{S}^{\star}$ is computed by the Floyd–Warshall recursion [73] in $O(n^3)$ time. The result is symmetric, never smaller than $\mathbf{S}$, keeps the unit diagonal, and is max-product transitive: $s^{\star}_{ik} \geq s^{\star}_{ij} s^{\star}_{jk}$ for all i, j, k. The classical alternative of fuzzy set theory is the max-min closure $\max_P \min_{(u,v)\in P} s_{uv}$ of Zadeh's similarity relations [74], which makes $1 - \mathbf{S}^{\star}$ an ultrametric and is therefore suited to hierarchical clustering rather than to embedding; it is available as `transitive_closure(S, "max-min")`.

The closure changes the picture in a specific way. Without it, all pairs without a common hyperedge are equally dissimilar (distance 1) and the embedding has to place them at the same distance, which is impossible in two dimensions for more than three such vertices and produces a crowded center. With it, distances grow with the number and weakness of the links in between, and the layout spreads the vertices along the chains of the hypergraph.

5.2.3 Embedding the dissimilarities

The dissimilarity matrix $\mathbf{\Delta} = \mathbf{1} - \mathbf{S}^{\star}$ (with zero diagonal) is embedded in the plane by *classical multidimensional scaling* [75]. With the matrix of squared dissimilarities $\mathbf{\Delta}^{(2)} = [\delta^2_{ik}]$ and the centering matrix $\mathbf{J} = \mathbf{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^{\mathsf{T}}$, the doubly centered matrix

$$\mathbf{G} = -\tfrac{1}{2}\,\mathbf{J}\,\mathbf{\Delta}^{(2)}\,\mathbf{J} \tag{5.3}$$

is the Gram matrix of a configuration that reproduces $\mathbf{\Delta}$ exactly whenever $\mathbf{\Delta}$ is Euclidean. Its two largest eigenpairs $(\lambda_1, \mathbf{u}_1), (\lambda_2, \mathbf{u}_2)$ give the coordinates $\mathbf{Y} = [\sqrt{\lambda_1}\mathbf{u}_1, \sqrt{\lambda_2}\mathbf{u}_2]$; negative eigenvalues, which occur for non-Euclidean dissimilarities, are discarded. Classical MDS is deterministic and fast, but it fits the large dissimilarities best and tends to compress the small ones, so groups of similar vertices may collapse.

Sammon mapping [76] weights each pair by the inverse of its dissimilarity and minimizes

$$E_{\text{Sammon}} = \frac{1}{\sum_{i<k} \delta_{ik}} \sum_{i<k} \frac{(\delta_{ik} - d_{ik})^2}{\delta_{ik}}, \tag{5.4}$$

where $d_{ik} = \|\mathbf{y}_i - \mathbf{y}_k\|$; small dissimilarities are therefore preserved more faithfully, and local neighborhoods are easier to read. `hatvis` minimizes (5.4) with Sammon's diagonal Newton step and step halving, starting from the classical MDS configuration; pairs with $\delta_{ik} = 0$ (equivalent vertices) are excluded from the sum and stay together.

5.2.4 Drawing on the similarity map

HAT-VIS draws every hyperedge as the *minimum spanning tree* of its members with respect to the dissimilarities $1 - s_{ik}$ of (5.1), i.e. along the strongest similarity links inside the group, colored per hyperedge and with a line width proportional to the hyperedge weight or to a hyperedge centrality; vertex sizes can show a vertex centrality. The trees use little ink and stay readable when many hyperedges overlap, which is typical of similarity maps, where the members of a hyperedge are by construction close together. An *ego view* draws only the hyperedges of one chosen vertex, and the *seriated similarity matrix*, $\mathbf{S}^\star$ with rows and columns in the barycentric order of section 3.2.2, complements the map with a view in which every pairwise value can be read. Any style of section 4.2.4 can be used instead of the trees.

Equivalent vertices are placed on the same point. HAT-VIS writes their labels together, which `hatvis` reproduces ("v_1, v_2" in fig. 5.2); alternatively a small jitter separates them, or `H.collapse_nodes()` replaces each class by one vertex whose size shows the class size.

5.2.5 The membership-aware layout

A similarity layout is faithful, not legible: a vertex that resembles the members of a hyperedge without belonging to it may land inside the hull of that hyperedge. On the wire-harness space of section 3.4, classical MDS produces 290 intrusions in 60 of the 70 hyperedges (table 5.1). The *membership-aware layout* of `hatvis` combines the stress of the similarity embedding with a geometric constraint derived from the hyperedges. It alternates two steps (algorithm 5.1).

The first is one iteration of *stress majorization* (SMACOF) [77, 78] toward the dissimilarities: with the current configuration $\mathbf{Y}$ and $d_{ik} = \|\mathbf{y}_i - \mathbf{y}_k\|$, the Guttman transform

$$\mathbf{y}_i \leftarrow \frac{1}{n} \sum_{k \neq i} \frac{\delta_{ik}}{d_{ik}} \left(\mathbf{y}_i - \mathbf{y}_k\right) + \frac{1}{n} \sum_{k} \mathbf{y}_k \tag{5.5}$$

decreases the raw stress $\sum_{i<k} (d_{ik} - \delta_{ik})^2$ monotonically. The second is a *projection*: every hyperedge e defines a disc with center $\mathbf{c}_e$, the centroid of its members, and radius $\rho_e = \max_{j \in e} \|\mathbf{y}_j - \mathbf{c}_e\| + \mu$ with a small margin μ; every non-member inside the disc is moved a fraction κ of the way toward its boundary along the ray from the center. The projection is the separation-constraint idea of constrained stress layouts [79] applied to the discs of the hyperedges. Neither step alone solves

Algorithm 5.1 Membership-aware layout

Input: hypergraph H; dissimilarities Δ (default: one minus the transitive weighted Jaccard similarity); initial layout $\mathbf{Y}^{(0)}$ (classical MDS or Sammon); strength κ, margin μ, number of iterations T, check interval c

Output: layout $\mathbf{Y}^{\text{best}}$

$\mathbf{Y} \leftarrow \mathbf{Y}^{(0)}$; $\mathbf{Y}^{\text{best}} \leftarrow \mathbf{Y}$; $I^{\text{best}} \leftarrow$ intrusions of $\mathbf{Y}$;
for $t = 1, \dots, T$ **do**
  SMACOF step (5.5);
  foreach *hyperedge* e **do**
    $\mathbf{c}_e \leftarrow$ centroid of the members; $\rho_e \leftarrow \max_{j \in e} \|\mathbf{y}_j - \mathbf{c}_e\| + \mu$;
    foreach *non-member* j *with* $\|\mathbf{y}_j - \mathbf{c}_e\| < \rho_e$ **do**
      move $\mathbf{y}_j$ by $\kappa\,(\rho_e - \|\mathbf{y}_j - \mathbf{c}_e\|)$ along $(\mathbf{y}_j - \mathbf{c}_e)/\|\mathbf{y}_j - \mathbf{c}_e\|$;
  if t *is a multiple of* c **then**
    $I \leftarrow$ number of (non-member, hyperedge) pairs inside the padded hulls;
    if $I \le I^{best}$ **then** $\mathbf{Y}^{\text{best}} \leftarrow \mathbf{Y}$, $I^{\text{best}} \leftarrow I$;

the problem: the majorization step pulls similar non-members back in, and the projection alone destroys the geometry. Alternating them, and keeping the iterate with the fewest intrusions counted on the actual padded hulls, yields a layout with nearly the stress of the similarity embedding and far fewer false enclosures. The fraction κ (`strength`, 0.25 by default) controls the trade-off.

5.2.6 Quality function deployment as a hypergraph

Quality function deployment (QFD) translates the *voice of the customer* into engineering specifications [80, 81]. Its central tool, the *house of quality* [82], is a matrix whose rows are customer requirements with importance ratings, whose columns are engineering specifications, and whose cells are relationship strengths on a 9/3/1 scale; a triangular roof records the correlations between specifications. Abonyi and Czvetkó [3] observed that this matrix is the incidence matrix of a hypergraph: every specification is a hyperedge that joins the requirements it affects, the incidence value is the relationship strength, and the importance of a requirement can be multiplied into its row. The HAT-VIS case study [1] uses exactly this construction, with

$$b_{ie} = r_{ie} \cdot u_i, \tag{5.6}$$

where $r_{ie} \in \{0, 1, 3, 9\}$ is the relationship between requirement i and specification e and u_i the importance of the requirement. The similarity map then shows

In a nutshell Similarity-based visualization (HAT-VIS)

- Profiles: weighted incidence rows $a_{ie} = w_e b_{ie}$; similarity: weighted Jaccard (5.1) (Dice, overlap, cosine, inclusion, Hamming available).
- Max-product transitive closure (5.2): strongest chain, Floyd–Warshall, $O(n^3)$.
- Embedding of $\mathbf{\Delta} = \mathbf{1} - \mathbf{S}^\star$: classical MDS (5.3) (global, deterministic) or Sammon mapping (5.4) (local neighborhoods).
- Drawing: minimum spanning trees of the members along the strongest similarity links, widths by weight or centrality, ego view, seriated similarity matrix; equivalent vertices coincide.
- Membership-aware layout: SMACOF step plus disc projection of non-members, keep the iterate with the fewest intrusions; parameter `strength`.
- QFD: requirements as vertices, specifications as hyperedges, $b_{ie} = r_{ie} u_i$.

which requirements are served by the same specifications, the dual map shows which specifications serve the same requirements (and are therefore candidates for integration or substitution), and centralities rank both. The optimization problems built on the same hypergraph, selecting a set of specifications under a budget and assigning specifications to requirements by network flow, are the subject of section 10.2.5.

5.3 Using it in `hatvis`

```
vertex_similarity(H, measure="jaccard", weighted=True,
transitive=False, closure="max-product", as_frame=False)
```

`measure` `"jaccard"`, `"dice"`, `"overlap"`, `"cosine"`, `"inclusion"`, `"hamming"`, `"count"`.
`weighted` scale the incidence rows by the hyperedge weights (HAT-VIS).
`transitive`, `closure` apply the max-product or max-min closure.
Related `edge_similarity(H)` (the same on the dual), `transitive_closure(S, method)`, `dissimilarity(H)`, `hypergraph_distance(H1, H2, kind)`.

```
layout_similarity(H, measure="jaccard", transitive=True,
method="cmds", weighted=True, jitter=0.0)
```

`method` `"cmds"` (classical MDS) or `"sammon"`.
`jitter` small random displacement that separates equivalent vertices.

Related `layout_membership(H, init="similarity", n_iter=300, strength=0.25, margin=0.04, check_every=10); classical_mds(D), sammon(D); draw_hatvis(H, mode="mst", ego=None, width_by, size_by); plot_similarity_matrix(S, order); datasets.load_electric_car_qfd()`.

5.4 Examples

The pipeline step by step. On the five-vertex example the four steps of HAT-VIS are four lines.

```
S = hv.vertex_similarity(H)                    # weighted Jaccard
ST = hv.transitive_closure(S, "max-product")   # strongest chains
Delta = 1 - ST
Y = hv.layout.classical_mds(Delta)             # 2-D configuration
```

Figure 5.2 shows the two similarity matrices and the drawing. The Jaccard similarity of v_1 and v_2 is 1 (equal memberships), that of v_1 and v_3 is $1/3$ (one shared hyperedge out of three), that of v_3 and v_5 is $1/2$, and v_1 and v_5 have similarity 0. The transitive closure raises the last value to $0.167 = \frac{1}{3} \cdot \frac{1}{2}$, the value reported in the HAT-VIS paper, and leaves all others unchanged, because they are already the strongest chains. Classical MDS places v_1 and v_2 on the same point $(-0.389, 0)$, v_3 and v_4 symmetrically at $(0.170, \pm 0.333)$, and v_5 at $(0.437, 0)$: the layout has the mirror symmetry of the hypergraph, which is invariant under exchanging v_3 with v_4 and e_1 with e_2.

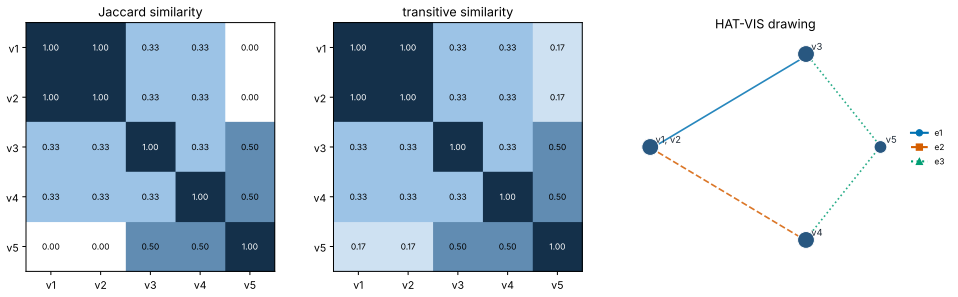


Figure 5.2. The HAT-VIS pipeline on the five-vertex example: weighted Jaccard similarity (left), its max-product transitive closure (middle), and the drawing on the classical MDS layout with hyperedges as minimum spanning trees (right). v_1 and v_2 have equal memberships and share a point.

An autonomous electric car. The house of quality of the HAT-VIS case study relates eight customer requirements to eleven engineering specifications. The requirements are autonomous driving quality and reliability (CR1, importance 5), fast battery charging (CR2, 4), short travel time (CR3, 3), obstacle avoidance

(CR4, 4), a user-friendly interface (CR5, 3), boot space (CR6, 2), a good braking system (CR7, 4), and an efficient safety system (CR8, 5). The specifications are LiDAR (ES1), GPS navigation (ES2), the battery management system (ES3), grid charging (ES4), sensors (ES5), algorithms (ES6), the classification of the vehicle (ES7), the number of seats (ES8), the type of brakes (ES9), airbags (ES10), and the supplementary restraint system (ES11). With the incidence (5.6), the requirements are the vertices and the specifications the hyperedges; the script ranks both with the linear node–edge eigenvector centrality of Tudisco and Higham [83], which is explained in section 6.2.5.

```
x, y = hv.node_edge_eigenvector(H, "linear")
print("vertex centrality:\n",
    x.sort_values(ascending=False).round(4).head(4).to_string())
print("hyperedge centrality:\n",
    y.sort_values(ascending=False).round(4).head(4).to_string())
```

Autonomous driving quality (CR1, centrality 0.230) and the safety system (CR8, 0.206) are the most central requirements, followed at a distance by braking (CR7, 0.126) and obstacle avoidance (CR4, 0.125). Among the specifications, sensors (ES5) and algorithms (ES6) are tied at the top (0.158), followed by LiDAR (ES1, 0.134) and the vehicle classification (ES7, 0.114). The ranking agrees with the reading of the house of quality: the perception and decision stack serves the two most important requirements with the strongest relationships.

```
fig, axes = plt.subplots(1, 3, figsize=(18, 5.6))
draw_hatvis(H, ax=axes[0], width_by=y, size_by=x, title="requirements
    (vertices) and specifications")
draw_hatvis(H, ax=axes[1], ego="CR4", width_by=y, size_by=x, title="ego
    view of CR4 (obstacle avoidance)")
draw_hatvis(H.dual(), ax=axes[2], title="dual: specifications as vertices")
```

Figure 5.3 shows the three views. On the similarity map autonomous driving (CR1) and obstacle avoidance (CR4) form the tightest pair (transitive similarity 0.38, the largest in the matrix), because both rely on LiDAR, sensors, and algorithms. Charging (CR2), travel time (CR3), and boot space (CR6) form a second group on the opposite side, held together by the battery management system, the vehicle classification, and the number of seats, while the interface (CR5), the safety system (CR8), and braking (CR7) spread between the two groups: each shares some specifications with both. The ego view of obstacle avoidance (CR4) shows that it is served by exactly three specifications, LiDAR, sensors, and algorithms, all of which also serve CR1. In the dual map the airbags (ES10) and the supplementary restraint system (ES11) coincide: they relate to exactly the same requirements with the same strengths, so from the point of view of the house of quality they are one specification, a fact that the matrix shows only to a careful reader and the map shows at once.

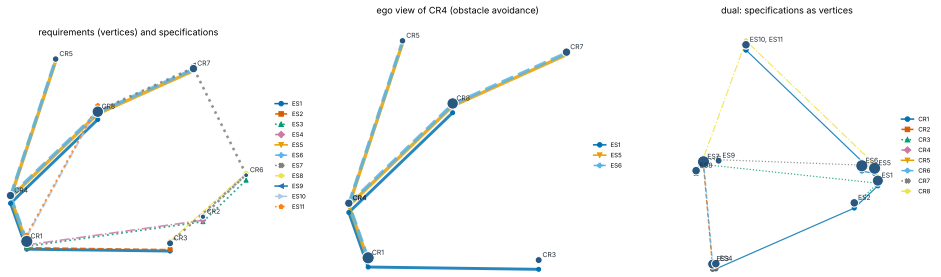


Figure 5.3. Similarity maps of the electric-car house of quality. Left: requirements as vertices and specifications as minimum-spanning-tree hyperedges, line widths by hyperedge centrality and vertex sizes by vertex centrality. Middle: the ego view of obstacle avoidance (CR4). Right: the dual map, in which airbags (ES10) and the supplementary restraint system (ES11) share a point.

Fewer intrusions. The last script compares classical MDS, Sammon mapping, and the membership-aware layout on the whole wire-harness space and on a fourteen-activity part of production line 1 (AGV loading, crimp positioning, handover, and two assembly activity types).

```
layouts = {"classical MDS": hv.layout_similarity(H, jitter=0.01,
random_state=0),
           "Sammon mapping": hv.layout_similarity(H, method="sammon"),
           "membership-aware": hv.layout_membership(H, random_state=0)}
quality = pd.DataFrame({name: hv.layout_quality(H, pos) for name, pos
in layouts.items()}).T
```

Table 5.1. Membership-aware layout compared with classical MDS and Sammon mapping on the whole wire-harness space (31 resources, 70 activities) and on a fourteen-activity part of line 1. Stress is measured against the transitive Jaccard dissimilarities.

Data	Layout	Stress	Intrusions	Hyperedges with intrusions
whole space	classical MDS	0.211	290	60
	Sammon mapping	0.137	83	47
	membership-aware	0.182	69	27
line-1 part	classical MDS	0.107	23	12
	Sammon mapping	0.085	4	3
	membership-aware	0.083	1	1

Table 5.1 shows the trade-off. On the whole space, classical MDS crowds the many resources that share the AGV and the sensors into a small region and produces 290 intrusions. Sammon mapping, which preserves small dissimilarities better, has the lowest stress and reduces the intrusions to 83. The membership-aware layout accepts a stress between the two and reduces the number of hyperedges

with an intrusion from 47 to 27. On the line-1 part the membership-aware layout is better on both measures, with a single intrusion left (fig. 5.4). On the production example of table 4.1 it had the lowest stress and no intrusion at all. The projection does not create the geometry, which still comes from the similarities; it only resolves the conflicts that the similarity embedding leaves, and it is most effective when the hypergraph is not too dense for a planar drawing.

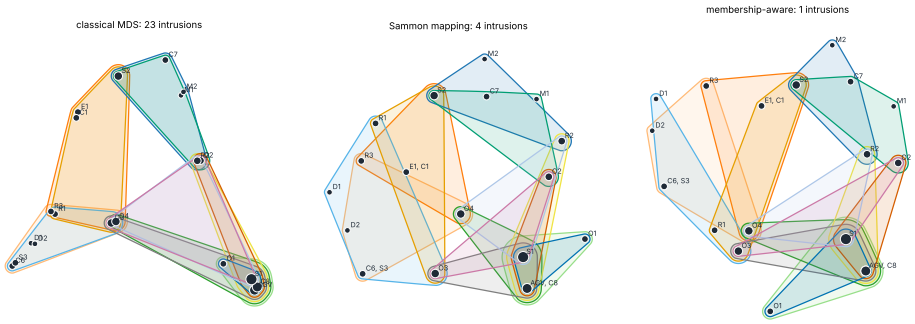


Figure 5.4. Classical MDS (left), Sammon mapping (middle), and the membership-aware layout (right) on a fourteen-activity part of production line 1, drawn with rounded hulls. The number of intrusions is given in each title.

5.5 Further reading

The HAT-VIS paper [1] describes the MATLAB implementation and the electric-car case study; the QFD hypergraph is introduced in [3]. Torgerson [75] and Kruskal [60] are the classical references for metric and non-metric scaling, Sammon [76] for the mapping that bears his name, and de Leeuw and Mair [77] for the majorization algorithm; Gansner, Koren, and North [78] brought stress majorization to graph drawing and Dwyer, Koren, and Marriott [79] added separation constraints. Zadeh [74] introduced similarity relations and their transitive closures. Surana, Chen, and Rajapakse [19] and Felippe, Kirkley, and Battiston [84] compare whole hypergraphs rather than vertices, a question taken up in chapter 12.

CHAPTER

6



Walks, connectivity, and centrality

Which hyperedges are strongly connected, which vertices hold a system together, and which elements would disconnect it if they failed? This chapter answers these questions with three families of tools. The *s*-walks of Aksoy et al. measure connectivity at a chosen strength of overlap and yield *s*-components, *s*-distances, and the *s*-betweenness and *s*-closeness centralities that Nagy et al. used to find the critical elements of a manufacturing space. Random walks on hypergraphs lead to stationary distributions and PageRank. Eigenvector centralities defined on the adjacency tensor and the nonlinear node–edge centralities of Tudisco and Higham make the importance of a group depend on all of its members, and the hyper-core decomposition identifies the densely interacting nucleus of a hypergraph. The examples compare these measures on the wire-harness space and on synthetic hypergraphs.

6.1 Problem and idea

In a graph, two edges either share a vertex or not, and a path is a sequence of edges that share vertices. In a hypergraph two hyperedges may share one vertex or ten, and the difference matters. Two production activities that share only the AGV are connected in a weak sense: material passes from one to the other. Two activities that share an operator, a robot, a tool, and a sensor are connected in a strong sense: they form a cell, and a disturbance in one propagates immediately to the other. A path through the hypergraph that uses only single shared vertices is a chain of weak contacts; a path that uses large overlaps is a chain of strong ones. A graph-based distance on the clique expansion or on the ordinary line graph cannot tell them apart.

The idea of Aksoy et al. [21] is to make the required overlap a parameter. An s-walk is a sequence of hyperedges in which consecutive hyperedges share at least s vertices (fig. 6.1). For $s = 1$ this is the usual notion of connectivity; as s grows, only strong overlaps count, the hypergraph falls apart into s-components, and distances and centralities computed at several values of s describe the system at several strengths of interaction. The same idea applies to vertices, which are s-adjacent when they share at least s hyperedges.

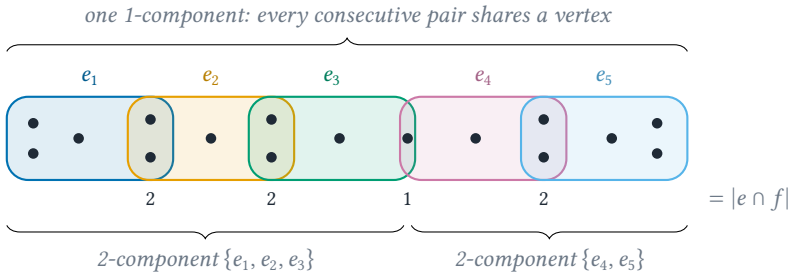


Figure 6.1. s-walks and s-components. Five hyperedges form a chain; consecutive hyperedges share two vertices except e_3 and e_4, which share one. All five form one 1-component, but the 2-walks stop at the weak link, and the chain splits into two 2-components. The weak link is invisible in the clique expansion, where every overlap produces the same kind of edge.

6.2 Theory

6.2.1 Degrees and distributions

The simplest centralities are local. The degree d_i of a vertex, weighted or not, counts its hyperedges; the *strength* of a hyperedge, $w_e \sum_i b_{ie}$, is its weighted size. Their distributions, together with the distribution of the overlaps $|e \cap f|$ of intersecting hyperedges, are the first fingerprint of a hypergraph (fig. 2.5); empirical hypergraphs typically have heavy-tailed degree distributions, a few typical hyperedge sizes, and overlaps concentrated on one or two vertices [29, 85].

6.2.2 s-walks, s-distances, and s-components

An *s-walk* of length ℓ is a sequence of hyperedges $e_0, e_1, \dots, e_\ell$ with $|e_{t-1} \cap e_t| \geq s$ for $t = 1, \dots, \ell$; equivalently, a walk in the s-line graph $L_s(H)$ of (2.3). The *s-distance* $d_s(e, f)$ is the length of the shortest s-walk from e to f (infinite if there is none), the *s-components* are the connected components of $L_s(H)$, and the *s-diameter* is the largest finite s-distance [21]. Only hyperedges of size at least s take part. The vertex versions use the s-line graph of the dual: vertices v_i and v_j are s-adjacent when $c_{ij} \geq s$ in (2.1).

The s-components are nested: every $(s+1)$-component lies inside an s-component, so increasing s refines the partition. `hatvis` summarizes the refinement in two ways. The *s-component profile* lists, for $s = 1, 2, \dots$, the number of elements that take part, the number of components, the size and diameter of the largest one, and the number of links of $L_s(H)$; it is a compact fingerprint of how strong the overlaps of a hypergraph are. The *s-profile plot* draws one column per value of s and one row per hyperedge, colored by the s-component the hyperedge belongs to, with the rows in an order that keeps components contiguous at every level; the plot reads like a dendrogram in which the merging height is the overlap strength.

6.2.3 s-centralities

Any centrality of the s-line graph is an s-centrality of the hypergraph [21]. Three are used here. The *s-betweenness* of a hyperedge is the share of shortest s-walks between other hyperedges that pass through it,

$$BC_s(e) = \sum_{f \neq e \neq g} \frac{\sigma^s_{fg}(e)}{\sigma^s_{fg}}, \tag{6.1}$$

with σ^s_{fg} the number of shortest s-walks from f to g and $\sigma^s_{fg}(e)$ the number of those through e [86], normalized by the number of pairs. The *s-closeness* is the reciprocal mean s-distance to the reachable elements. Because a hypergraph usually becomes disconnected for $s \geq 2$, the *harmonic s-closeness*

$$HCC_s(e) = \frac{1}{|E_s| - 1} \sum_{f \in E_s,\, f \neq e} \frac{1}{d_s(e, f)}, \qquad E_s = \{e : |e| \geq s\}, \tag{6.2}$$

is preferable: unreachable elements contribute zero instead of making the measure undefined [87]. The *s-eigenvector centrality* is the principal eigenvector of the weighted adjacency of $L_s(H)$. Applied to the dual, all three rank vertices.

Nagy et al. [4] proposed to rank the elements of a manufacturing space by averaging an s-centrality over the strengths of interaction,

$$\overline{C}_{s_{\max}}(v) = \frac{1}{s_{\max}} \sum_{s=1}^{s_{\max}} C_s(v), \tag{6.3}$$

with C the s-betweenness or the harmonic s-closeness. An element that is central only at $s = 1$ is a broker of weak contacts (a shared transport resource); an element that remains central at $s = 2$ and $s = 3$ is embedded in strongly overlapping activities (a shared competence of a cell). The average rewards both, and the individual values tell them apart. Algorithm 6.1 summarizes the computation.

Algorithm 6.1 *s-centralities of vertices or hyperedges*

Input: hypergraph H; overlap levels $s = 1, \dots, s_{\max}$; target: vertices or hyperedges
Output: BC_s, HCC_s for each s and their averages (6.3)

if *the target is the vertices* **then** replace H by its dual H^*;
$\mathbf{K} \leftarrow \mathbf{B}^\top\mathbf{B}$ (overlap sizes);
for $s = 1, \dots, s_{\max}$ **do**
    build $L_s(H)$: nodes $\{e : |e| \geq s\}$, links $\{e, f\}$ with $k_{ef} \geq s$;
    $BC_s \leftarrow$ betweenness on $L_s(H)$ (Brandes' algorithm) (6.1);
    $HCC_s \leftarrow$ harmonic closeness on $L_s(H)$ (6.2); elements with $|e| < s$ get 0;
average over s (6.3);

6.2.4 Random walks and PageRank

A random walk on a hypergraph moves between vertices through hyperedges. In the walk of Zhou et al. [35], a walker at vertex u chooses one of its hyperedges with probability proportional to the weight, w_e/d_u, and then a vertex of that hyperedge uniformly at random. The transition matrix is

$$\mathbf{P} = \mathbf{D}_v^{-1}\mathbf{B}\mathbf{W}\mathbf{D}_e^{-1}\mathbf{B}^\top, \tag{6.4}$$

the operator of the random-walk Laplacian (2.5). The walker may stay at its vertex (it can choose itself in the second step), and on a connected hypergraph the stationary distribution is proportional to the weighted degree, $\pi_i = d_i/\sum_j d_j$. Two generalizations matter in practice. Chitra and Raphael [40] let the second step depend on *edge-dependent vertex weights* $\gamma_e(v)$, so that a vertex can be more important in one hyperedge than in another (an author's contribution to a paper, the relationship score of a requirement in a QFD hyperedge); they showed that with edge-independent weights every hypergraph walk is equivalent to a walk on a weighted clique expansion, so that only edge-dependent weights add genuinely higher-order information. Carletti et al. [88] weight the step from u to v by $\sum_{e \ni u,v}(|e| - 1)$, which favors moves inside large hyperedges. `hatvis` provides all of them and the simple walk on the clique expansion.

The *PageRank* [89] of a hypergraph walk is the stationary distribution of the walk that follows $\mathbf{P}$ with probability α and teleports to a random vertex (or to a personalization distribution) with probability $1 - \alpha$. With the Zhou walk on an unweighted hypergraph, PageRank is close to the normalized degree; with edge-dependent weights it can differ substantially, because a vertex with a high weight in a few important hyperedges collects more probability than a vertex with low weights in many.

6.2.5 Eigenvector centralities

An eigenvector centrality declares a vertex important if it is connected to important vertices. On a graph the definition is the Perron vector of the adjacency matrix; on a hypergraph there are several choices, and they differ in how the importance of the co-members of a hyperedge is combined [83, 90].

Clique-expansion eigenvector centrality (CEC). The Perron vector of the clique adjacency $\mathbf{A}$ of (2.2), $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$, adds the importance of the co-members: in a hyperedge $\{u, v, w\}$, vertex u receives $x_v + x_w$.

Z- and H-eigenvector centralities (ZEC, HEC). Benson [90] used the adjacency tensor of a k-uniform hypergraph (2.8). The tensor–vector product (2.9) *multiplies* the importance of the co-members, so in $\{u, v, w\}$ vertex u receives $x_v x_w$, and a hyperedge contributes only if all its other members are important. The Z-eigenvector centrality solves

$$\mathcal{A}\mathbf{x}^{k-1} = \lambda\mathbf{x}, \qquad \|\mathbf{x}\|_1 = 1, \quad \mathbf{x} > 0, \tag{6.5}$$

and the H-eigenvector centrality solves

$$\mathcal{A}\mathbf{x}^{k-1} = \lambda\mathbf{x}^{[k-1]}, \qquad \mathbf{x} > 0, \tag{6.6}$$

where $\mathbf{x}^{[k-1]}$ is the entrywise power. The H-eigenvector of a connected hypergraph is unique by the Perron–Frobenius theory of nonnegative tensors and is found by the iteration of Ng, Qi, and Zhou [91], $\mathbf{x} \leftarrow (\mathcal{A}\mathbf{x}^{k-1})^{[1/(k-1)]}$, normalized. Z-eigenvectors need not be unique; `hatvis` computes one with the shifted symmetric higher-order power method of Kolda and Mayo [92], $\mathbf{x} \leftarrow (\mathcal{A}\mathbf{x}^{k-1} + \alpha\mathbf{x})/\,\|\cdot\|_1$, from the uniform start. Both are defined for uniform hypergraphs; `hatvis` evaluates (2.9) from the incidence lists without forming the tensor.

Nonlinear node and edge centralities. Tudisco and Higham [83] rank vertices and hyperedges simultaneously. With positive vectors $\mathbf{x} \in \mathbb{R}^n$ (vertices) and $\mathbf{y} \in \mathbb{R}^m$ (hyperedges) and four positive maps f, g, φ, ψ applied entrywise, they look for

$$\lambda\,\mathbf{x} = g\big(\mathbf{B}\mathbf{W}f(\mathbf{y})\big), \qquad \mu\,\mathbf{y} = \psi\big(\mathbf{B}^{\mathsf{T}}\varphi(\mathbf{x})\big). \tag{6.7}$$

A vertex is important if it belongs to important hyperedges, and a hyperedge is important if it contains important vertices; the maps decide how. With all maps the identity, (6.7) is the pair of leading singular vectors of $\mathbf{B}\mathbf{W}^{1/2}$, the hypergraph analogue of the hubs and authorities of HITS (`kind="linear"`). With $\varphi = \ln$ and $\psi = \exp$ the importance of a hyperedge is the geometric mean of its members, so a single unimportant member pulls the whole hyperedge down (`"log-exp"`); with $\varphi(x) = x^p$, $\psi(x) = x^{1/p}$ and large p it approaches the maximum, so a hyperedge is as important as its most important member (`"max"`). Tudisco and Higham prove that, under conditions on the homogeneity degrees of the maps, the positive solution of (6.7) is unique and a nonlinear power iteration converges to it;

In a nutshell Connectivity and centrality

- s-walks: consecutive hyperedges share at least s vertices; s-distance, s-components (nested in s), s-diameter; s-component profile and s-profile plot.
- s-centralities: betweenness (6.1), harmonic closeness (6.2), eigenvector of $L_s(H)$; averages over s (6.3) (Nagy et al.); vertices via the dual.
- Random walks: Zhou (6.4) ($\pi \propto d$), edge-dependent (Chitra–Raphael), Carletti; PageRank.
- Eigenvector family: CEC (sum of co-members), ZEC and HEC (product of co-members, uniform hypergraphs), nonlinear node–edge (6.7) (linear, log-exp, max).
- Hyper-cores: (k, m)-cores and hyper-coreness (6.8).

`hatvis` uses the damped form $\mathbf{x} \leftarrow \sqrt{\mathbf{x} \odot g(\mathbf{B}\mathbf{W} f(\mathbf{y}))}$, $\mathbf{y} \leftarrow \sqrt{\mathbf{y} \odot \psi(\mathbf{B}^\top \varphi(\mathbf{x}))}$ with normalization. The linear version was used for the electric-car house of quality in section 5.4, where the vertex and the hyperedge centralities rank the requirements and the specifications at the same time.

6.2.6 Hyper-cores

The k-core of a graph is its largest subgraph in which every vertex has degree at least k. Mancastroppa et al. [93] generalized it to hypergraphs with a second parameter that controls the size of the interactions. The (k, m)-*hyper-core* is the maximal subhypergraph in which every vertex belongs to at least k hyperedges that have at least m members *inside the core*; it is found by repeatedly removing vertices that violate the condition, shrinking the hyperedges accordingly, and discarding hyperedges with fewer than m remaining members. The m-core number $C_m(i)$ is the largest k for which vertex i belongs to the (k, m)-hyper-core, and the *hyper-coreness*

$$R(i) = \sum_{m=2}^{M} \frac{C_m(i)}{k_{\max}(m)}, \qquad k_{\max}(m) = \max_j C_m(j), \tag{6.8}$$

combines the core numbers over all interaction sizes. Mancastroppa et al. showed that higher-order processes such as threshold contagion localize in the hyper-cores and that seeding a process in the vertices of highest hyper-coreness is more efficient than seeding by degree.

6.3 Using it in `hatvis`

`s_betweenness(H, s=1, edges=True, normalized=True)`

edges `True` ranks hyperedges, `False` ranks vertices (through the dual).

Related `s_closeness`, `s_harmonic_closeness`, `s_eigenvector`, `average_s_centrality(H, func, s_max, edges)`, `s_centrality_table(H, s_values, edges)`; `s_connected_components`, `s_distance`, `s_distance_matrix`, `s_diameter`, `s_component_profile`; plots `plot_s_profile`, `plot_s_walks`, `plot_ranking`.

`node_edge_eigenvector(H, kind="linear", p=10)`

kind `"linear"`, `"log-exp"`, `"max"`; returns the vertex and the hyperedge centralities as two `Series`.

Related `degree_centrality`, `edge_strength`, `clique_eigenvector`, `z_eigenvector`, `h_eigenvector`, `pagerank(H, alpha=0.85, edge_dependent=False)`, `hypercore_numbers(H, m)`, `hypercoreness(H)`; `spectral.transition_matrix(H, kind)`, `spectral.stationary_distribution`.

6.4 Examples

How strongly are the activities connected? The s-component profile of the wire-harness space is computed for hyperedges and for vertices.

```
print(hv.s_component_profile(H, s_values=range(1, 5)))
print(hv.s_component_profile(H, s_values=range(1, 5), edges=False))
```

Table 6.1. s-component profile of the wire-harness space: number of participating elements (size or degree at least s), number of s-components, size and diameter of the largest component, and number of links of the s-line graph.

	activities (hyperedges)					resources (vertices)				
s	elem.	comp.	largest	diam.	links	elem.	comp.	largest	diam.	links
1	70	1	70	2	1031	31	1	31	3	141
2	70	4	24	3	253	29	1	29	4	73
3	56	17	12	1	143	28	1	28	5	47
4	40	29	4	1	14	28	1	28	5	35

Table 6.1 shows two very different behaviors. The activities are all 1-connected, with diameter 2: any two activities are linked through at most one intermediate

activity, because the sensors and the AGV touch almost everything. At $s = 2$ they split into four components, the largest with 24 activities, and at $s = 3$ into seventeen, so the strong overlaps are local: the stations are cells, connected to each other only through single shared resources. The resources, in contrast, stay connected for all four values of s while their diameter grows from 3 to 5: they are linked by chains of resources that share several activities, but these chains become long. The s-profile plot of fig. 6.2 shows the refinement for every activity; the vertex s-line graphs of fig. 6.3 show how the links between resources thin out from $s = 1$ to $s = 2$ while the sensors and the crimping competence become the brokers of the remaining strong links.

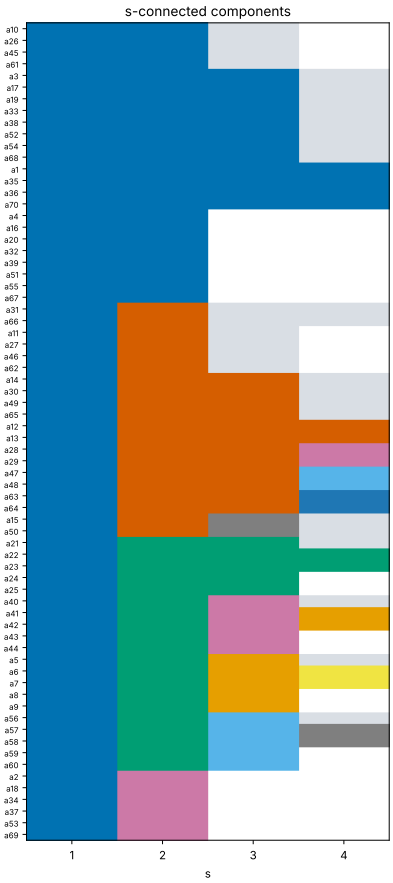


Figure 6.2. s-profile of the 70 activities of the wire-harness space for $s = 1, \dots, 4$. Each row is an activity, each column a value of s; the color identifies the s-component, and white means that the activity is smaller than s or isolated at that level.

Critical elements of the manufacturing space. Following Nagy et al. [4], the resources are ranked by s-betweenness and harmonic s-closeness for $s = 1, 2, 3$ and by their averages.

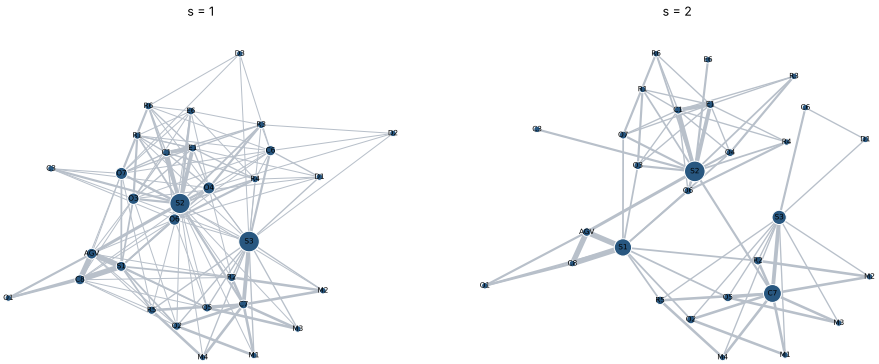


Figure 6.3. Vertex s-line graphs of the wire-harness resources for $s = 1$ (left) and $s = 2$ (right): two resources are linked if they share at least s activities. Vertex sizes show the betweenness in the line graph, edge widths the number of shared activities.

```
table = hv.s_centrality_table(H, s_values=(1, 2, 3), edges=False)
avg_bc = hv.average_s_centrality(H, hv.s_betweenness, s_max=3, edges=False)
avg_hc = hv.average_s_centrality(H, hv.s_harmonic_closeness, s_max=3,
    edges=False)
```

At $s = 1$ the two sensors S_3 and S_2 have by far the largest betweenness (0.185 and 0.177), followed by the operators of the assembly stations; the handling sensor S_1, although it takes part in 24 activities, has a small betweenness at this level, because it is adjacent to almost every other resource and therefore rarely needed as an intermediary. At $s = 2$ the ranking changes: S_2 rises to 0.395, and the crimping competence C_7 jumps from rank 11 to rank 2 (0.287), followed by S_1 (0.266). The competence is not a broker of weak contacts; it is the element through which the strongly overlapping crimping activities of the four stations are connected. The averages over $s = 1, 2, 3$ rank S_2, C_7, S_1, and S_3 highest by betweenness and S_2, S_1, C_7, S_3 by harmonic closeness (fig. 6.4). The result agrees with the conclusion of the original study that the information and competence elements, not the physical machines, hold the collaborative space together; it also shows why the analysis must be done at several values of s, since C_7 would have been missed at $s = 1$.

A family of centralities. The eight vertex centralities of sections 6.2.4 to 6.2.6 are compared on a 3-uniform Chung–Lu hypergraph (section 9.2.1) with 40 vertices of heterogeneous expected degree and 70 hyperedges.

```
C = pd.DataFrame({
    "degree": hv.degree_centrality(H),
    "CEC": hv.clique_eigenvector(H),
    "ZEC": hv.z_eigenvector(H),
    "HEC": hv.h_eigenvector(H),
    "linear NEC": hv.node_edge_eigenvector(H, "linear")[0],
    "max NEC": hv.node_edge_eigenvector(H, "max")[0],
    "PageRank": hv.pagerank(H),
```

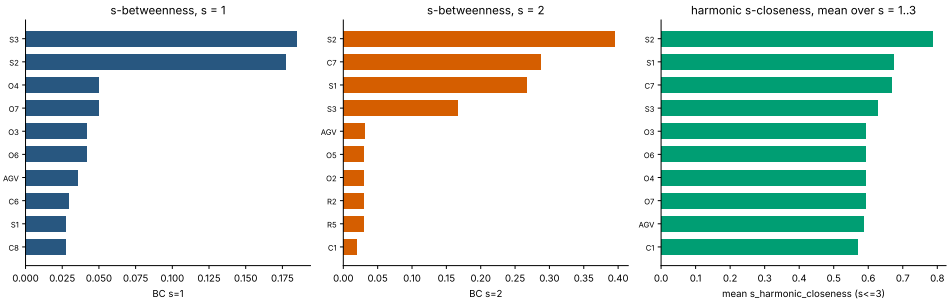


Figure 6.4. Critical elements of the wire-harness space: s-betweenness of the resources for $s = 1$ (left) and $s = 2$ (middle), and harmonic s-closeness averaged over $s = 1, 2, 3$ (right). The crimping competence C_7 is marginal at $s = 1$ and second at $s = 2$.

```
    "hyper-coreness": hv.hypercoreness(H),
})
corr = C.corr(method="spearman")
```

The Spearman correlations in fig. 6.5 show three groups. Degree, PageRank, and hyper-coreness agree almost perfectly (degree–PageRank 0.99, degree–hyper-coreness 0.97): on a uniform hypergraph the Zhou walk has the degree as stationary distribution, and PageRank only adds teleportation. CEC, HEC, and the two node–edge centralities form a second group (pairwise correlations 0.93–0.99). ZEC stands apart: its correlation with PageRank is only 0.70 and with degree 0.77. Because ZEC multiplies the importance of the co-members, a vertex of moderate degree whose hyperedges contain other central vertices can outrank a vertex of high degree whose hyperedges are formed with peripheral ones. Which centrality is "right" depends on the process: if a hyperedge contributes only when all its members act (a team that needs everyone, a reaction that needs all reactants), the multiplicative ZEC or the log-exp node–edge centrality is the appropriate model; if any member can transmit (gossip in a group), CEC or PageRank is.

Hyper-cores. The last example plants a nucleus: eight core vertices $c_0, \dots, c_7$ form 25 hyperedges of three to five members among themselves, and 60 further hyperedges of two or three members are drawn from the core and forty peripheral vertices $p_0, \dots, p_{39}$.

```
cores = pd.DataFrame({f"m={m}": hv.hypercore_numbers(H, m) for m in
    range(2, 6)})
R = hv.hypercoreness(H)
```

The core numbers separate the two groups at every interaction size (fig. 6.6). For $m = 2$ most core vertices have core number 14 (two of them 9 and 10), and the peripheral vertices at most 4; for $m = 5$ only the core vertices remain, with core number 4, because no peripheral vertex takes part in hyperedges of five members. The hyper-coreness ranks exactly the eight core vertices at the top. A degree ranking would mix in peripheral vertices that happen to take part in many small

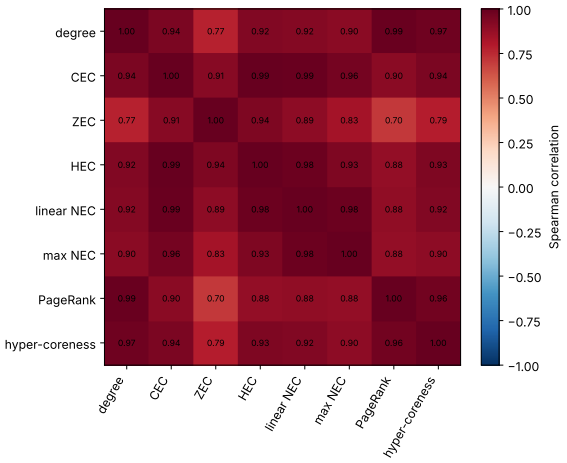


Figure 6.5. Spearman correlations of eight vertex centralities on a 3-uniform Chung–Lu hypergraph with 40 vertices and 70 hyperedges.

hyperedges; the hyper-core decomposition distinguishes many small interactions from fewer large ones.

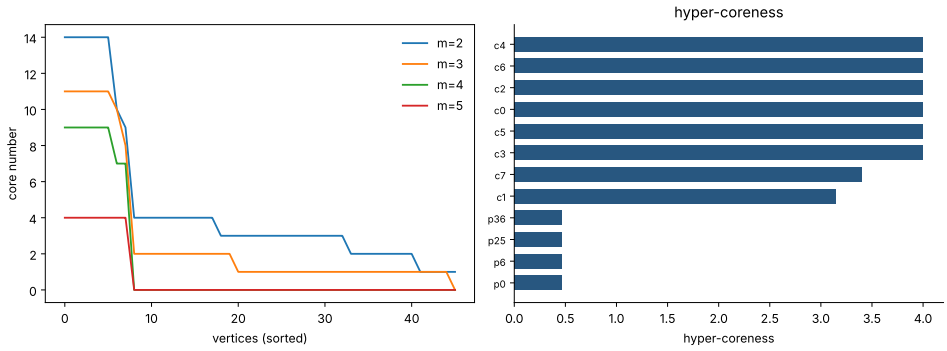


Figure 6.6. Hyper-core numbers of all vertices for $m = 2, \ldots, 5$, sorted (left), and the twelve vertices of largest hyper-coreness (right) in a hypergraph with a planted core of eight vertices.

6.5 Further reading

Aksoy et al. [21] develop the s-walk framework with many applications; HyperNetX [20] implements it. Nagy et al. [4] apply s-centralities to the analysis and design of manufacturing spaces. Benson [90] compares the clique, Z-, and H-eigenvector centralities, and Tudisco and Higham [83] introduce the nonlinear node and edge centralities with their existence and uniqueness theory. Random walks on hypergraphs are studied by Zhou et al. [35], Chitra and Raphael [40], Carletti et al. [88], and Hayashi et al. [39]. Mancastroppa et al. [93] introduce

hyper-cores and their role in higher-order processes. Estrada and Rodríguez-Velázquez [94] defined early subgraph centralities and clustering coefficients for hypernetworks.

CHAPTER

7



Clustering and partitioning

Grouping the vertices of a hypergraph, its hyperedges, or both at once is the most common analysis after centrality: communities of co-authors, cells of a manufacturing system, narratives of a news collection, blocks of a circuit to be placed on separate chips. This chapter explains why a hypergraph objective counts a split group differently from a graph objective, defines the strict and all-or-nothing hypergraph modularities and a Louvain-type heuristic that maximizes them, and presents the alternatives available in `hatvis`: Louvain on projections, label propagation through the incidences, spectral clustering with the hypergraph Laplacian, and spectral co-clustering of vertices and hyperedges. The second half treats balanced partitioning with the cut-net and connectivity objectives and the Fiduccia–Mattheyses refinement. The examples recover planted communities, form manufacturing cells, and separate the narratives of the news data.

7.1 Problem and idea

A community is a set of vertices that interact more among themselves than with the rest. For a graph, "interact" means "share an edge"; for a hypergraph, it means "share a hyperedge", and the question is how a hyperedge that is split between two communities should be counted. Consider a meeting of four people, two from each of two departments (fig. 7.1). In the clique expansion the meeting becomes six pairs, four of which cross the department boundary, so the split costs four. As a hyperedge it is one meeting that involves both departments, so the split costs one. Larger groups make the difference larger: a hyperedge of size k split evenly contributes $k^2/4$ crossing pairs but one crossing hyperedge. A graph objective therefore penalizes splitting a large group much more than splitting a small one, while a hypergraph objective treats every group as a unit, whatever its size.

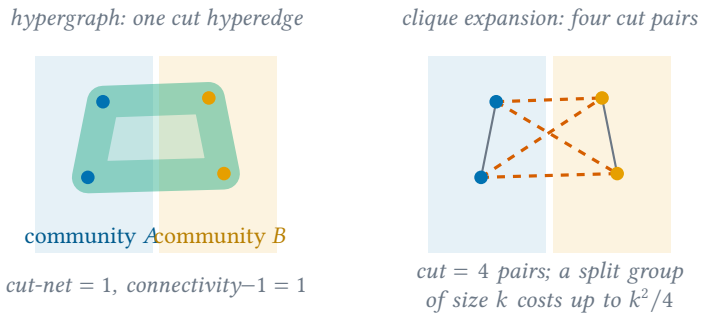


Figure 7.1. A group of four vertices split evenly between two communities. As a hyperedge it is one crossing interaction (left); in the clique expansion it becomes four crossing pairs (right, dashed). Graph objectives therefore weight the splitting of large groups quadratically, hypergraph objectives linearly or not at all.

Neither convention is right in general. If the cost of a split is a communication cost per pair (every pair of people who must coordinate across a department boundary), the clique expansion is the correct model; if the cost is per group (a meeting that needs a room in both buildings, a net that must be routed between two chips, a task that requires resources of two cells), the hypergraph objective is. The idea of this chapter is to make the choice explicit: every method below states which representation and which counting it uses, and `hatvis` provides methods of both kinds so that the reader can choose by the cost model rather than by the software at hand [95].

7.2 Theory

7.2.1 Modularity

The modularity of a partition $\boldsymbol{\sigma}$ of a graph compares the weight of the edges inside the communities with the weight expected in a random graph with the same degrees [96]:

$$Q_G = \frac{1}{2M} \sum_{i,j} \left(a_{ij} - \gamma \frac{k_i k_j}{2M} \right) [\sigma_i = \sigma_j], \tag{7.1}$$

with degrees k_i, total weight M, and resolution γ. Applied to the clique expansion (2.2), it is the standard way of clustering a hypergraph with graph tools, and it inherits the quadratic weighting of large hyperedges described above.

Kamiński et al. [97] defined modularity directly on the hypergraph. In their *strict* version a hyperedge counts as internal only if all its vertices are in the same community, and the expected number of internal hyperedges of size k is computed from a random hypergraph that preserves the degrees, in which the probability

that a hyperedge of size k falls entirely into community A_c is $(\operatorname{vol} A_c / \operatorname{vol} V)^k$, with $\operatorname{vol} A = \sum_{i \in A} d_i$:

$$Q_H = \frac{1}{W} \sum_{k \geq 2} \Big[\sum_{e \in E_k} w_e \, [e \subseteq A_\sigma] \; - \; \gamma \, W_k \sum_c \Big(\frac{\operatorname{vol} A_c}{\operatorname{vol} V} \Big)^k \Big], \tag{7.2}$$

where E_k are the hyperedges of size k, W_k their total weight, $W = \sum_k W_k$, and $[e \subseteq A_\sigma]$ indicates that e lies inside one community. For a graph ($k = 2$ only) the expression reduces to the usual modularity up to the treatment of self-loops. Chodrow, Veldt, and Benson [98] derived an *all-or-nothing* modularity of the same structure as the maximum-likelihood objective of a degree-corrected hypergraph stochastic block model in which a hyperedge is more likely when all its members are in one community; in their objective each size k has its own weight, estimated from the data, which lets the model learn whether large groups tend to be homogeneous. Both objectives are *all-or-nothing*: a hyperedge with one member outside the community earns nothing. This is the right model for interactions that require the whole group, and it is the reason why these objectives prefer coarse partitions that keep large hyperedges intact.

7.2.2 Maximizing hypergraph modularity

The Louvain method [99] maximizes graph modularity by moving single vertices to the neighboring community with the largest gain and then aggregating communities into super-vertices. Under the strict objective (7.2) single-vertex moves from a singleton start rarely pay off, because a hyperedge becomes internal only when its last member joins, so a naive transfer of the method stalls near the singleton partition. The heuristic of `hatvis` (algorithm 7.1) therefore starts from the Louvain partition of the clique expansion, which already groups vertices that share many hyperedges, then alternates exact local moves and greedy merges of communities, both evaluated with the hypergraph objective, until neither improves it. The merge step follows the agglomerative scheme considered for hypergraph modularity by Kamiński et al. [97]; Kumar et al. [100] proposed a different route, iteratively reweighting the clique expansion so that graph Louvain approximates the hypergraph objective.

7.2.3 Other community methods

Louvain on projections. Louvain on the clique expansion groups vertices; Louvain on the weighted s-line graph groups hyperedges, which is how Nagy et al. [4] found activity-based modules of a manufacturing space in addition to resource-based ones. Both are fast and robust; they optimize graph modularity of the chosen projection, not (7.2).

Label propagation. The label propagation algorithm of Raghavan et al. [101] lets every vertex adopt the most frequent label among its neighbors. On a hypergraph

Algorithm 7.1 Hypergraph Louvain (strict modularity)

Input: hypergraph H; resolution γ; initialization (clique Louvain or singletons)
Output: partition $\boldsymbol{\sigma}$

$\boldsymbol{\sigma} \leftarrow$ Louvain partition of the clique expansion (or singletons);
repeat
  `// local moving`
  foreach *vertex v in random order* **do**
    for every community c of a neighbor of v, compute the exact change ΔQ_H of (7.2) if v moves to c;
    if *the best* $\Delta Q_H > 0$ **then** move v;
  `// greedy merging`
  repeat
    merge the pair of communities sharing a hyperedge whose union increases Q_H most;
  until *no merge increases* Q_H;
until $\boldsymbol{\sigma}$ *does not change*;

`hatvis` propagates through the incidences: every hyperedge adopts the weighted majority label of its members, then every vertex adopts the weighted majority label of its hyperedges, with weights $w_e b_{ie}$ and random tie-breaking, until the labels are stable. It needs no parameter and runs in linear time per sweep, but it is sensitive to noise and tends to merge communities when many hyperedges cross them.

Spectral clustering. Zhou et al. [35] generalized the normalized cut to hypergraphs. For a vertex set S with complement S^c,

$$\operatorname{Ncut}(S) = \operatorname{vol}\partial S\Big(\frac{1}{\operatorname{vol} S} + \frac{1}{\operatorname{vol} S^c}\Big), \qquad \operatorname{vol}\partial S = \sum_e w_e \frac{|e \cap S|\,|e \cap S^c|}{|e|}, \tag{7.3}$$

where a hyperedge contributes in proportion to the number of member pairs it splits, normalized by its size. The relaxed problem is solved by the eigenvectors of the normalized Laplacian (2.4); for k communities the rows of the k leading eigenvectors of $\mathbf{D}_v^{-1/2}\mathbf{B}\mathbf{W}\mathbf{D}_e^{-1}\mathbf{B}^\top\mathbf{D}_v^{-1/2}$ are normalized to unit length and clustered with k-means [102]. Spectral clustering needs the number of communities and fails on disconnected hypergraphs (each component produces a zero eigenvalue), but it is deterministic up to the k-means start and uses global information.

Co-clustering of vertices and hyperedges. Spectral co-clustering [51] (section 3.2.5) assigns vertices and hyperedges jointly to k groups by bipartite spectral partitioning of the incidence matrix (3.5). It answers a different question from the

vertex methods: not only which vertices belong together, but also which hyperedges define each group. For a document collection the co-clusters are topics with their documents, for a manufacturing space resource cells with their activities.

Statistical models. Generative models infer communities with uncertainty and allow overlapping memberships. Beyond the stochastic block model behind the all-or-nothing modularity [98], Contisciani, Battiston, and De Bacco [103] and Ruggeri et al. [104] fit mixed-membership models that scale to very large hypergraphs; Hypergraphx [23] implements several of them, and the converters of section 2.3 make them available for `hatvis` hypergraphs.

7.2.4 Balanced partitioning

Partitioning asks for k blocks of nearly equal size that cut as little as possible. It is the classical hypergraph problem of VLSI design, where vertices are cells, hyperedges are nets, and blocks are chips or regions, and it is used in parallel computing to distribute data with little communication [105, 106]. For a partition into blocks $V_1, \dots, V_k$, let λ_e be the number of blocks that hyperedge e touches. Three cost functions are standard:

$$\text{cut-net} = \sum_{e:\lambda_e>1} w_e, \qquad \text{connectivity}{-}1 = \sum_{e} w_e(\lambda_e - 1), \qquad \text{SOED} = \sum_{e:\lambda_e>1} w_e\lambda_e. \tag{7.4}$$

The cut-net counts every split hyperedge once; the connectivity metric counts how many additional blocks each hyperedge reaches, which is the communication volume when every block holding a member must receive the data of the net; the sum of external degrees (SOED) is the sum of the two. For $k = 2$ all three coincide up to a factor. The balance constraint requires $|V_j| \le (1+\varepsilon)\lceil n/k \rceil$. Finding an optimal balanced partition is NP-hard [107].

The *Fiduccia–Mattheyses* (FM) heuristic [108] improves a bisection by moving single vertices between the two blocks. In each pass every vertex is moved exactly once, always the vertex whose move gives the largest decrease of the cut among those allowed by the balance constraint, even if the gain is negative; the gains are updated incrementally from the number of members each hyperedge has in each block. At the end of the pass the sequence of moves is rolled back to the prefix with the smallest cut. Accepting negative gains inside a pass lets FM escape local minima, and the gain buckets make a pass run in time linear in the number of incidences. Multilevel partitioners such as hMETIS [105] and KaHyPar [106] coarsen the hypergraph, partition the coarsest level, and refine with FM on the way back; `hatvis` implements the flat multi-start FM (algorithm 7.2), sufficient for hypergraphs up to tens of thousands of incidences, and obtains k-way partitions by recursive bisection.

The normalized cut (7.3) offers a different balance: it does not fix the block sizes but divides the cut by the volumes, so that cutting off a small piece is expensive.

Algorithm 7.2 Fiduccia–Mattheyses bisection

Input: hypergraph H; imbalance ε; number of random starts
Output: balanced bisection with the smallest cut-net found

```
foreach random start do
    draw a random balanced bisection; for every hyperedge count its
      members in each block;
    repeat
        unlock all vertices; compute the gain of moving each vertex
          (cut hyperedges that become uncut minus uncut hyperedges
          that become cut);
        while an unlocked vertex can move without violating the
          balance do
            move the unlocked vertex of largest gain; lock it; update
              the gains of the vertices of its hyperedges;
        roll back to the prefix of moves with the smallest cut (among
          balanced states);
    until a pass does not improve the cut;
return the best bisection over all starts
```

The sign of the Fiedler vector of the normalized Laplacian is its spectral relaxation for $k = 2$ [35].

7.2.5 Comparing partitions

When a ground truth is known, as in the synthetic examples, partitions are compared with the *adjusted Rand index* (ARI) of Hubert and Arabie [109]: the fraction of vertex pairs on which two partitions agree (together in both or apart in both), corrected for the agreement expected by chance. It is 1 for identical partitions, close to 0 for independent ones, and does not depend on the labels or the number of communities.

7.3 Using it in `hatvis`

```
hypergraph_louvain(H, resolution=1.0, init="clique", random_
state=None, max_rounds=20)
```

Returns a partition (`Series` of community labels $0, \dots, k-1$).
Related `modularity(H, partition, resolution)` (strict hypergraph modularity), `graph_modularity(H, partition)`, `louvain(H, target="nodes"|"edges", weighting, resolution, s)`,

In a nutshell **Clustering and partitioning**

- A split hyperedge of size k costs one unit in a hypergraph objective and up to $k^2/4$ pairs in the clique expansion; choose by the cost model.
- Strict hypergraph modularity (7.2) (Kamiński et al.) and all-or-nothing modularity (Chodrow et al.): a hyperedge is internal only if all members are in one community; hypergraph Louvain: clique-Louvain start, exact local moves, greedy merges.
- Alternatives: Louvain on the clique expansion (vertices) or line graph (hyperedges), label propagation through incidences, spectral clustering with the normalized hypergraph cut (7.3), spectral co-clustering of vertices and hyperedges.
- Partitioning: cut-net, connectivity−1, SOED (7.4) under a balance constraint; multi-start Fiduccia–Mattheyses, recursive bisection; remove isolated vertices before spectral bisection.
- Evaluation: modularity, cut metrics, adjusted Rand index.

`label_propagation(H)`, `spectral_clustering(H, n_clusters)`, `spectral_coclustering(H, n_clusters)`, `communities(partition)`, `partition_sizes(partition)`.

`partition.fm_bisection(H, imbalance=0.03, n_starts=10, max_passes=20, random_state=None)`

Returns a two-block partition.

Related `partition.recursive_bisection(H, n_blocks)`, `partition.cut_metrics(H, partition)` (cut-net, connectivity−1, SOED, number of blocks, imbalance), `partition.normalized_cut(H, subset)`, `spectral.fiedler_vector(H)`.

7.4 Examples

Planted communities. A hypergraph stochastic block model (section 9.2.1) with 120 vertices in four equal communities and 300 hyperedges of sizes 2 to 5 is generated for decreasing shares p_{in} of intra-community hyperedges; the remaining hyperedges draw their members from the whole vertex set. Four methods are compared by the ARI against the planted partition.

```
methods = {
    "hypergraph Louvain": hv.hypergraph_louvain(H, random_state=0),
    "clique Louvain": hv.louvain(H, random_state=0),
    "spectral (Zhou)": hv.spectral_clustering(H, 4, random_state=0),
    "label propagation": hv.label_propagation(H, random_state=0),
}
```

Table 7.1. Adjusted Rand index of four community detection methods on a planted-partition hypergraph (120 vertices, four communities, 300 hyperedges of sizes 2–5) as the share of intra-community hyperedges p_{in} decreases.

p_{in}	hypergraph Louvain	clique Louvain	spectral (Zhou)	label propagation
0.90	1.000	1.000	1.000	1.000
0.75	1.000	1.000	1.000	0.978
0.60	1.000	1.000	0.978	0.548
0.45	0.956	0.872	0.869	0.380
0.30	0.144	0.206	0.190	0.000

All methods recover the communities when most hyperedges are internal (table 7.1, fig. 7.2). Label propagation degrades first: at $p_{\text{in}} = 0.6$ it has merged communities, and at 0.3 it returns a single one. The interesting regime is $p_{\text{in}} = 0.45$, where more than half of the hyperedges are random: hypergraph Louvain still reaches an ARI of 0.956, while clique Louvain and spectral clustering fall to about 0.87. In this model the planted structure is defined by whole hyperedges being internal, which is exactly what the strict modularity counts, whereas the clique expansion dilutes it with the many crossing pairs of the large random hyperedges. At $p_{\text{in}} = 0.3$ no method finds the structure; the three modularity- and spectrum-based methods are equally weak there, and the strict objective, which needs internal hyperedges to reward a community, is slightly worse than the clique expansion.

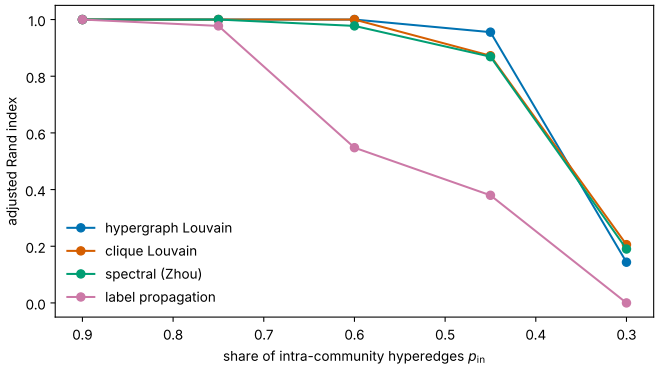


Figure 7.2. Adjusted Rand index of four community detection methods against the planted partition as the share of intra-community hyperedges decreases (values in table 7.1).

Forming manufacturing cells. Following Nagy et al. [4], the activities of the wire-harness space are grouped by Louvain on their line graph and the resources by hypergraph Louvain.

```
act = hv.louvain(H, target="edges", random_state=0)
res = hv.hypergraph_louvain(H, random_state=0)
res_clique = hv.louvain(H, random_state=0)
```

The activities fall into three communities of 24, 24, and 22. The first contains all transport and handling between stations (buffer handling, hand-overs, and most AGV loading); the second the twenty crimping activities together with four assembly activities; the third the remaining sixteen assembly activities with six AGV loading steps. Each community contains activities of both lines in equal numbers, so the modules are functional, not line-based: the two lines are copies of each other, and what distinguishes activities is the kind of work, not where it happens. The resources form three communities: the transport pair $\{O_1, C_8\}$ (the operator and the competence of AGV loading); a community of fifteen resources with all four crimping machines, the crimping competence C_7, the robots R_2 and R_5, the operators O_2 and O_5, the three screwdrivers with the screw-fastening competence C_6, and the sensors S_1 and S_3; and a community of fourteen resources with the other operators and robots, the AGV, the tools E_1 and E_6, the competences C_1 and C_3, and the sensor S_2. The strict modularity of the hypergraph Louvain partition is 0.419, higher than the 0.394 of the clique Louvain partition evaluated with the same objective. Figure 7.3 shows the incidence matrix ordered by the two partitions, in which the cells appear as blocks, and the clique expansion colored by the resource communities. The AGV itself does not stay with the transport pair: it takes part in more activities than any operator, and the objective places it with the resources with which it shares the largest volume of activities. The partition is a proposal, not a design; it shows which groupings keep the most activities inside one cell and which resources (here the sensors and the screwdrivers) would have to be shared between cells in any division.

Balanced partitioning. A planted two-block hypergraph with 80 vertices and 200 hyperedges of sizes 2 to 4 ($p_{\text{in}} = 0.85$) is bisected by multi-start FM, by the sign of the Fiedler vector, and at random.

```
fm = partition.fm_bisection(H, imbalance=0.05, n_starts=10, random_state=0)
Hc = H.remove_isolated()                          # an isolated vertex
    would get its own block
fiedler = (hv.spectral.fiedler_vector(Hc) >
    0).astype(int).reindex(H.nodes, fill_value=0)
rnd = pd.Series(np.random.default_rng(0).permutation([0, 1] * 40),
    index=H.nodes)
table = pd.DataFrame({name: partition.cut_metrics(H, p) for name, p in
                      {"FM": fm, "Fiedler sign": fiedler, "random": rnd,
    "planted": truth}.items()}).T
table["normalized cut"] = [partition.normalized_cut(H, p.index[p == 0])
    for p in (fm, fiedler, rnd, truth)]
```

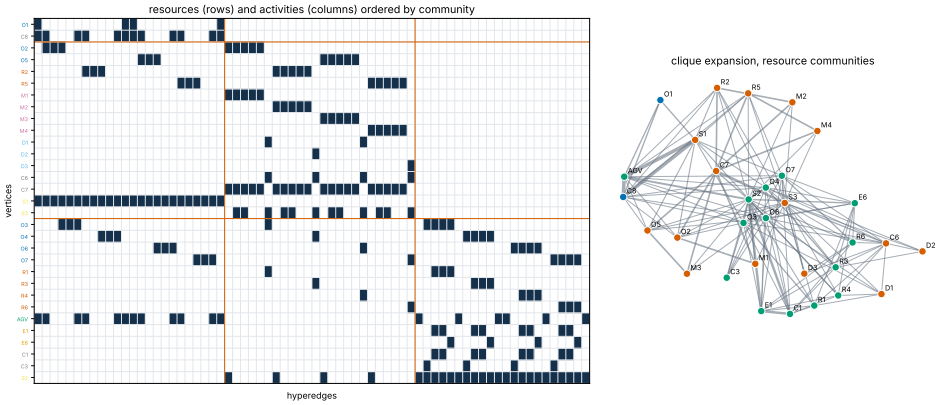


Figure 7.3. Manufacturing cells of the wire-harness space. Left: the incidence matrix with resources and activities ordered by their communities (orange lines separate the communities). Right: the clique expansion of the resources colored by the hypergraph Louvain communities.

FM and the Fiedler bisection both find the planted partition exactly: cut-net and connectivity−1 of 25, normalized cut 0.121, perfect balance; the random bisection cuts 146 hyperedges. The hypergraph has one isolated vertex, and the script removes it before computing the Fiedler vector. Without that step the normalized Laplacian has a second zero eigenvalue, its eigenvector is concentrated on the isolated vertex, and the sign split puts one vertex in one block and the other 79 in the other: a cut of zero with an imbalance of 0.975, a degenerate solution that the normalized cut avoids only when the hypergraph is connected. FM, which enforces the balance explicitly, is immune to this. Recursive bisection into four blocks cuts 108 hyperedges with connectivity−1 of 115 at 5% imbalance; since only two blocks are planted, the second level has to cut through communities, and the cost rises accordingly.

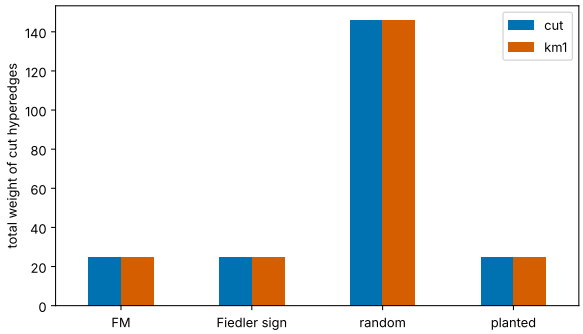


Figure 7.4. Cut-net and connectivity−1 of four bisections of a planted two-block hypergraph: multi-start Fiduccia–Mattheyses, the sign of the Fiedler vector, a random balanced bisection, and the planted partition.

Co-clustering the news data. Spectral co-clustering of the incidence matrix of 180 synthetic articles (60 per period) into six co-clusters groups themes with the articles that use them.

```
rl, cl = hv.spectral_coclustering(T, n_clusters=6, random_state=0)
rows = list(rl.sort_values(kind="stable").index)
cols = list(cl.sort_values(kind="stable").index)
```

The cross-tabulation of the theme co-clusters against the planted narratives is a near-permutation: climate science, policy, society, and energy each form a co-cluster of exactly their four themes, and the finance and impacts themes share one co-cluster. The sixth co-cluster has no themes of its own; its 24 articles are mostly mixtures of the climate narrative with another one, which the method sets apart from the single-narrative articles. Ordering the incidence matrix by the co-clusters (fig. 7.5) shows the narratives as dense blocks, with the two-narrative articles as the off-diagonal entries.

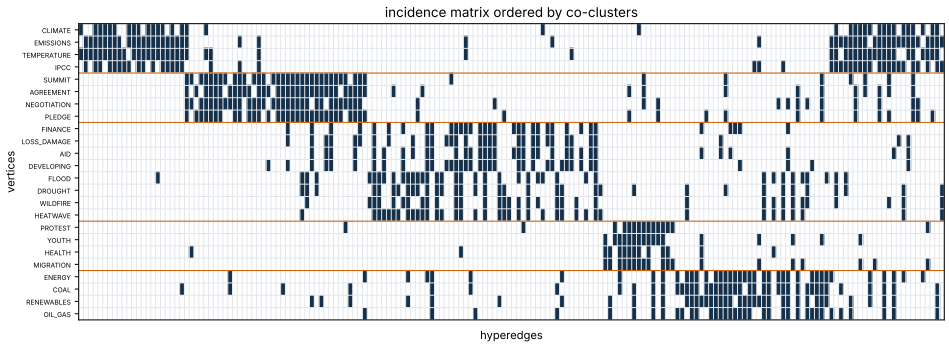


Figure 7.5. The incidence matrix of 180 synthetic news articles (columns) and 24 themes (rows) ordered by spectral co-clustering; orange lines separate the theme co-clusters.

7.5 Further reading

Kamiński et al. [97] define hypergraph modularity and discuss its strict and weaker variants; Chodrow, Veldt, and Benson [98] derive the all-or-nothing modularity from a stochastic block model and give a Louvain algorithm for it; Kumar et al. [100] approximate it by reweighting the clique expansion. Veldt, Benson, and Kleinberg [95] unify hypergraph cut objectives through splitting functions and show when they can be reduced to graph cuts. Contisciani et al. [103] and Ruggeri et al. [104] develop scalable mixed-membership models. For partitioning, Karypis et al. [105] describe the multilevel paradigm of hMETIS and Schlag et al. [106] the state-of-the-art KaHyPar framework.

CHAPTER

8



Frequent patterns and association rules

A transaction database and a hypergraph are the same object seen from two sides: every transaction is a set of items, every hyperedge a set of vertices. This chapter uses the correspondence in both directions. Frequent itemsets mined from the hyperedges are themselves the hyperedges of a weighted pattern hypergraph, closed and maximal itemsets compress it, and the inclusion lattice orders it; association rules become hyperarcs of a directed hypergraph, and the weights of association-rule hypergraph partitioning turn the patterns into clusters of mutually predictive items. Because frequent is not the same as significant, the chapter ends with a statistical test that keeps only the groups that occur more often than the activity of their members explains. The examples use the synthetic news data with planted narratives and a hypergraph with planted repeated groups.

8.1 Problem and idea

Market-basket analysis, the mining of co-occurring symptoms, the search for recurring teams, and the analysis of themes in news articles share one question: which combinations of items occur together often, and which items predict others? The classical answer is frequent-itemset mining [110]: count the transactions that contain each candidate set and keep the sets whose relative frequency exceeds a threshold. The result is a list, often thousands of sets long, sorted by support. It is hard to read because the sets overlap and nest: if $\{a, b, c\}$ is frequent, so are $\{a, b\}$, $\{a, c\}$, $\{b, c\}$, and the three singletons.

The idea of this chapter is to treat the patterns as a hypergraph and to apply the tools of the previous chapters to it (fig. 8.1). Every frequent itemset is a

hyperedge weighted by its support; the drawings of chapter 4, the UpSet view of section 3.2.6, and the clustering methods of chapter 7 then show and group the patterns. Association rules $A \Rightarrow C$ are naturally directed hyperarcs from the antecedent set to the consequent set. In the other direction, any hypergraph can be mined: its hyperedges are the transactions, and its frequent itemsets are the vertex groups that recur across hyperedges, the higher-order counterpart of a pair of vertices with a large co-occurrence count. Since the most active vertices co-occur often by chance alone, a frequent group must be tested against a null model before it is interpreted.

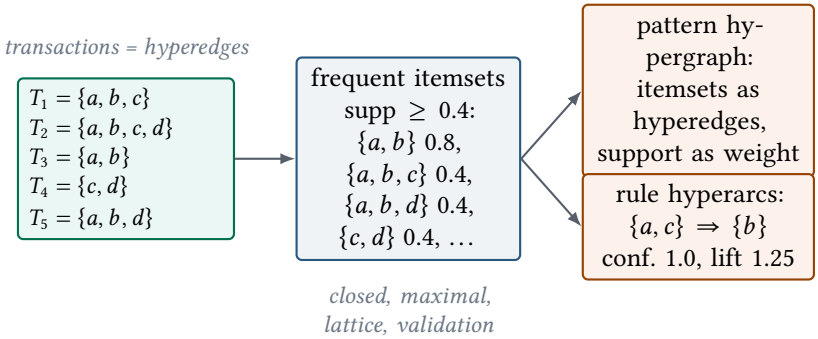


Figure 8.1. Patterns as hypergraphs. Five transactions (equivalently, five hyperedges) yield frequent itemsets with their supports; the itemsets are the hyperedges of a weighted pattern hypergraph, and association rules are hyperarcs from an antecedent set to a consequent set. In the example $\{a, b, c\}$ occurs in two of five transactions and $\{a, c\}$ in two, so the rule $\{a, c\} \Rightarrow \{b\}$ has confidence 1, and since $\text{supp}\{b\} = 0.8$ its lift is 1.25.

8.2 Theory

8.2.1 Support and the itemset lattice

Let $\mathcal{T} = (T_1, \ldots, T_N)$ be a database of transactions over an item set V, with optional weights w_T; for a hypergraph, the transactions are the hyperedges and the weights the hyperedge weights, so that a collapsed multi-hyperedge counts with its multiplicity. The *support* of an itemset $X \subseteq V$ is

$$\text{supp}(X) = \frac{\sum_{T \supseteq X} w_T}{\sum_T w_T}, \tag{8.1}$$

and X is *frequent* if $\text{supp}(X) \geq \sigma_{\min}$. Support is anti-monotone: $X \subseteq Y$ implies $\text{supp}(X) \geq \text{supp}(Y)$, so every subset of a frequent itemset is frequent, and every superset of an infrequent itemset is infrequent. This *downward closure* property makes the search over the $2^{|V|}$ subsets feasible.

Apriori. Agrawal and Srikant [110] generate the frequent itemsets level by level: the candidates of size $k+1$ are the unions of two frequent k-itemsets that share

Algorithm 8.1 Eclat (depth-first frequent itemset mining)

Input: transactions with weights; minimum support $\sigma_{\min}$; maximum size
Output: all frequent itemsets with their supports

```
1 procedure Extend(X, t(X), candidates C)
2     foreach item i ∈ C in a fixed order do
3         t ← t(X) ∩ t({i}); s ← Σ_{T∈t} w_T / Σ_T w_T;
4         if s ≥ σ_min then
5             output X ∪ {i} with support s;
6             Extend(X ∪ {i}, t, items of C after i);
7 Extend(∅, all transactions, frequent single items);
```

$k-1$ items, a candidate is pruned if any of its k-subsets is infrequent, and the survivors are counted in one pass over the database.

Eclat. Zaki [111] stores for every itemset its *tid-list*, the set of transactions that contain it, and explores the lattice depth first: the tid-list of $X \cup \{i\}$ is the intersection of the tid-lists of X and $\{i\}$, and its size is the support. Eclat needs no candidate counting passes and is usually faster than Apriori on dense data; both produce the same itemsets (algorithm 8.1).

Closed and maximal itemsets. The frequent itemsets are redundant. An itemset is *closed* if no proper superset has the same support, and *maximal* if no proper superset is frequent [112]. The closed itemsets determine the supports of all frequent itemsets (the support of X is the largest support of a closed superset), so they are a lossless compression; the maximal itemsets determine only which sets are frequent. The closed itemsets have a second meaning: they are the *intents* of the formal concepts of the incidence relation [113]. A formal concept is a pair (extent, intent) of a set of hyperedges and a set of vertices such that the intent is the set of vertices shared by all hyperedges of the extent and the extent is the set of all hyperedges that contain the intent. The concepts, ordered by inclusion of their extents, form the concept lattice, and the frequent closed itemsets are the intents of the concepts with large extents.

The inclusion lattice. The itemsets, ordered by inclusion, form a partially ordered set. Its Hasse diagram has an arc from every set to each of its covers (minimal proper supersets) and is drawn in levels by size. Drawn for the frequent itemsets of one group of items, it shows at a glance how the support decays as items are added and which combinations are missing.

8.2.2 Association rules as hyperarcs

An association rule $A \Rightarrow C$ with disjoint itemsets A (antecedent) and C (consequent) states that transactions containing A tend to contain C. Its support is $\operatorname{supp}(A \cup C)$ and its strength is measured by several interestingness measures [114]:

$$\begin{aligned} \operatorname{conf} &= \frac{\operatorname{supp}(A \cup C)}{\operatorname{supp}(A)}, \quad \operatorname{lift} = \frac{\operatorname{conf}}{\operatorname{supp}(C)}, \quad \operatorname{conviction} = \frac{1 - \operatorname{supp}(C)}{1 - \operatorname{conf}}, \\ \operatorname{leverage} &= \operatorname{supp}(A \cup C) - \operatorname{supp}(A)\,\operatorname{supp}(C), \\ \operatorname{Zhang} &= \frac{\operatorname{leverage}}{\max\left\{ \operatorname{supp}(AC)(1 - \operatorname{supp}(A)),\ \operatorname{supp}(A)\big(\operatorname{supp}(C) - \operatorname{supp}(AC)\big) \right\}}, \end{aligned} \tag{8.2}$$

with $\operatorname{supp}(AC) = \operatorname{supp}(A \cup C)$. Confidence is the conditional probability of C given A; lift compares it with the unconditional probability, so lift above 1 means a positive association; leverage is the difference between the observed and the expected joint support; conviction is large when the rule rarely fails. Zhang's measure [115] lies in $[-1, 1]$, is positive for association and negative for dissociation, and is less sensitive than lift to the support of the consequent. A rule is a *hyperarc* from the tail A to the head C, so a set of rules is a directed hypergraph (section 2.2.1); its weights can be any of the measures above, and its drawings and walks follow the general machinery of directed hypergraphs, including the reachability analysis of section 10.2.4.

8.2.3 Association-rule hypergraph partitioning

Han, Karypis, Kumar, and Mobasher [116, 117] proposed to cluster items through their association rules. Every frequent itemset X of size at least two becomes a hyperedge whose weight is the average confidence of the rules that predict one of its items from the others,

$$w(X) = \frac{1}{|X|} \sum_{i \in X} \operatorname{conf}\big(X \setminus \{i\} \Rightarrow \{i\}\big), \tag{8.3}$$

and the weighted hypergraph is partitioned so that the cut weight is small. Items in the same part predict each other strongly; items in different parts are associated only weakly. The original work used the multilevel partitioner hMETIS; `hatvis` computes the weights with `arhp_weights` and clusters the weighted itemset hypergraph with the hypergraph Louvain method of section 7.2.2, which does not need the number of clusters.

8.2.4 Statistically validated hyperedges

A frequent group may simply consist of very active vertices. Musciotto, Battiston, and Mantegna [118] test every observed group against a null model that preserves

In a nutshell Patterns as hypergraphs

- Hyperedges are transactions and transactions are hyperedges; support (8.1) is anti-monotone.
- Apriori (level-wise) and Eclat (depth-first, tid-lists) give the same frequent itemsets; closed itemsets compress losslessly and are the intents of formal concepts; maximal itemsets bound the frequent family; the inclusion lattice orders it.
- Rules $A \Rightarrow C$ are hyperarcs; confidence, lift, leverage, conviction, Zhang (8.2).
- ARHP: weight frequent itemsets by mean rule confidence (8.3), cluster the weighted hypergraph.
- Validation: iterated hypergeometric null (8.4) per hyperedge size, Bonferroni or FDR; frequent is not significant.

the activity of each vertex and destroys everything else. The hyperedges of each size k are treated separately: among the N_k hyperedges of size k, vertex i appears $d_i^{(k)}$ times. Under the null hypothesis each vertex chooses its $d_i^{(k)}$ hyperedges uniformly at random among the N_k, independently of the other vertices. For a group $g = \{i_1, \dots, i_k\}$ the number X_g of hyperedges that contain all its members is then the size of the intersection of k independent random subsets of sizes $d_{i_1}^{(k)}, \dots, d_{i_k}^{(k)}$ of an N_k-element set. Its distribution is computed iteratively: the intersection of the first two subsets is hypergeometric, and each further subset intersects the current intersection hypergeometrically,

$$P\big(X^{(r+1)} = x\big) = \sum_y P\big(X^{(r)} = y\big) \frac{\binom{y}{x}\binom{N_k - y}{d_{i_{r+1}} - x}}{\binom{N_k}{d_{i_{r+1}}}}, \tag{8.4}$$

starting from $X^{(1)} = d_{i_1}$. A group observed n_{obs} times receives the p-value $P(X_g \geq n_{\text{obs}})$ and is *validated* if the p-value is below the significance level after correction for multiple testing. The conservative Bonferroni correction divides the level by the number of groups that could have been tested, $\binom{n_k}{k}$ for the n_k vertices active at size k; the Benjamini–Hochberg procedure [119] controls the false discovery rate over the observed groups instead and is less strict. Validation separates two situations that frequency confounds: a group of very active vertices that meets a few times by chance, and a group of moderately active vertices that meets more often than their activity explains.

8.3 Using it in `hatvis`

`frequent_itemsets(data, min_support=0.1, max_size=None, algorithm="eclat", weights=None, min_size=1)`

`data` a hypergraph (its hyperedges are the transactions) or a list of transactions.

`min_support` relative if at most 1, otherwise an absolute (weighted) count.

Returns a `DataFrame` with columns `itemset` (frozenset), `size`, `count`, `support`, sorted by decreasing support.

Related `closed_itemsets`, `maximal_itemsets`, `association_rules(itemsets, min_confidence)`, `itemset_hypergraph(itemsets, min_size)`, `patterns.rule_hypergraph(rules)`, `patterns.arhp_weights(itemsets)`, `patterns.inclusion_lattice(sets)`, `patterns.formal_concepts(H)`, `patterns.validated_hyperedges(H, alpha, correction)`, `transactions_to_hypergraph`, `hypergraph_to_transactions`; plots `plot_upset`, `plot_lattice`; SPMF input and output in `hatvis.io`.

8.4 Examples

Frequent theme combinations. With 150 synthetic articles per period (450 in total; the data are described in section 3.4), all theme combinations with a support of at least 5% are mined.

```
fi = hv.frequent_itemsets(T, min_support=0.05)
same = fi.equals(hv.frequent_itemsets(T, min_support=0.05,
    algorithm="apriori"))
closed = hv.closed_itemsets(fi)
maximal = hv.maximal_itemsets(fi)
IH = hv.itemset_hypergraph(closed[closed["support"] >= 0.08], min_size=2)
```

There are 88 frequent itemsets, Eclat and Apriori agree, and all 88 are closed: in these data no frequent combination always occurs together with an additional theme, so closure does not compress. The eight maximal itemsets are much more informative. The five largest are exactly the four-theme cores of five planted narratives (climate science with support 0.096, finance and impacts with 0.082 each, policy with 0.080, and energy with 0.073), and the sixth is a three-theme subset of the society narrative (YOUTH, MIGRATION, PROTEST, 0.056), whose fourth theme does not reach the threshold together with the other three. The pattern hypergraph of the closed itemsets with support of at least 8% (fig. 8.2) shows each narrative as a nest of overlapping hyperedges, the four-theme itemset enclosing

its triples and pairs; the absence of hyperedges between the nests is the visual signature of themes that rarely combine across narratives at this support level.

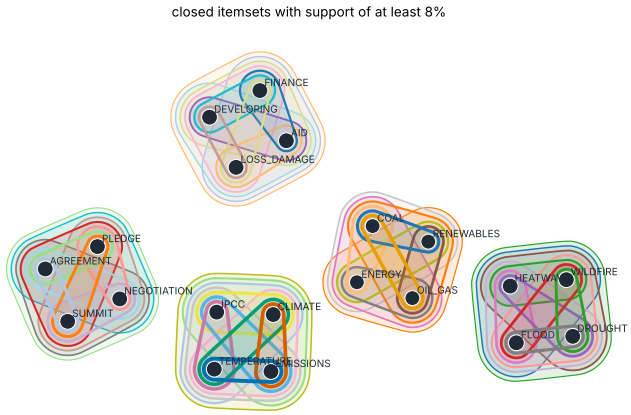


Figure 8.2. The pattern hypergraph of the synthetic news data: closed itemsets with support of at least 8% drawn as rounded hulls on the force-directed layout, line widths by support. Each narrative appears as a nest of itemsets.

The inclusion lattice of the frequent itemsets of the climate narrative (fig. 8.3) has all four singletons, all six pairs, all four triples, and the full set, which means that every combination of the four themes is frequent; the vertex sizes show how the support decreases level by level.

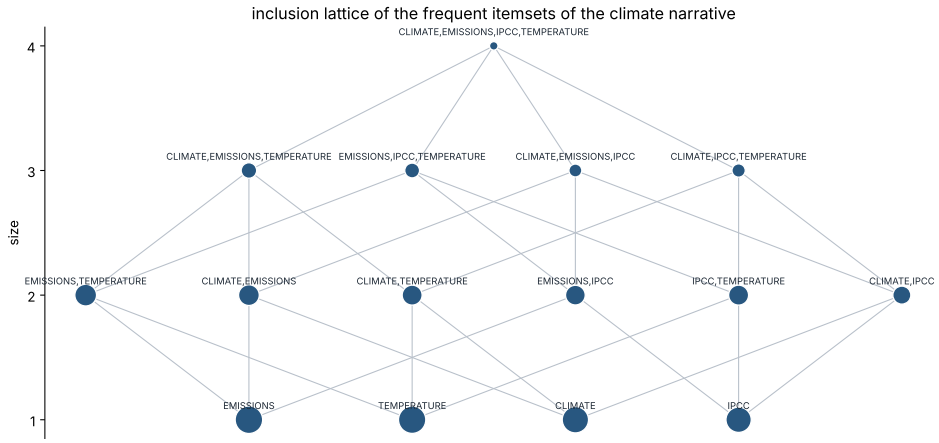


Figure 8.3. Inclusion lattice (Hasse diagram) of the frequent itemsets of the climate-science narrative, levels by size, vertex sizes by support.

Rules and clusters. The rules with confidence at least 0.6 are mined from the itemsets with support of at least 4%, the top thirty become a directed hypergraph, and the ARHP weights are used to cluster the itemset hypergraph.

```
rules = hv.association_rules(fi, min_confidence=0.6)
D = hv.patterns.rule_hypergraph(rules.head(30))
w = hv.patterns.arhp_weights(fi)
W = hv.itemset_hypergraph(fi,
    min_size=2).with_weights(w.reindex(fi[fi["size"] >=
    2]["itemset"]).to_numpy())
clusters = hv.hypergraph_louvain(W, random_state=0)
```

The strongest rule is {FINANCE, LOSS_DAMAGE, AID} ⇒ {DEVELOPING} with confidence 0.925 and lift 5.34: articles that discuss finance, loss and damage, and aid almost always mention developing countries, and they do so 5.3 times more often than articles in general. The following rules predict the fourth theme of a narrative from the other three (drought from wildfire and heatwave, confidence 0.90; summit from agreement, negotiation, and pledge, 0.88; coal from energy, oil and gas, and renewables, 0.87). The thirty strongest rules form a directed hypergraph with 30 hyperarcs on the 24 themes. The scatter plot of fig. 8.4 shows the usual trade-off: the rules with the highest lift have small support, because they involve three-theme antecedents. ARHP clustering of the itemset hypergraph returns six clusters that coincide exactly with the six planted narratives, including the society narrative, which the maximal itemsets had captured only partially: the clustering uses the pairwise and triple associations of the fourth theme even where the full four-theme combination is not frequent.

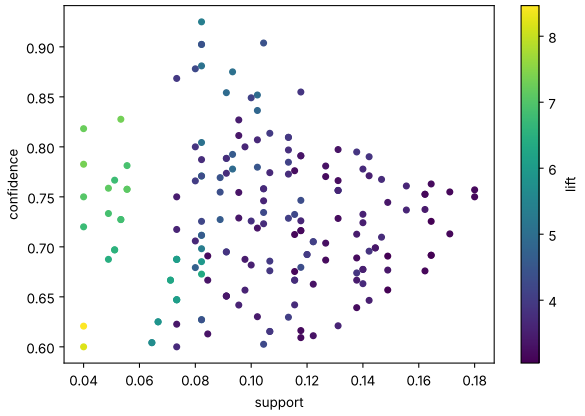


Figure 8.4. Support and confidence of the association rules of the news data with confidence at least 0.6, colored by lift.

Frequent versus significant. The last example plants four groups, $\{n_1, n_2, n_3\}$, $\{n_4, n_5, n_6\}$, $\{n_7, n_8\}$, and $\{n_1, n_9, n_{10}, n_{11}\}$, each repeated six times, among 300 random hyperedges of two to four of forty vertices, and validates every observed vertex set.

```
table = hv.patterns.validated_hyperedges(H, alpha=0.01,
    correction="bonferroni")
```

Exactly the four planted groups are validated (fig. 8.5). The four-vertex group $\{n_1, n_9, n_{10}, n_{11}\}$ has the smallest p-value (8.7×10^{-14}; its expected multiplicity under the null is 0.042, and Bonferroni counts $\binom{40}{4} = 91\,390$ possible tests), followed by the two triples and the pair (9.0×10^{-6} against an expected multiplicity of 0.88 and $\binom{39}{2} = 741$ tests). Groups that occur twice by chance have p-values around 10^{-3}, far above the Bonferroni threshold. The order of significance is not the order of frequency, since all planted groups occur six times: a repeated pair is much less surprising than a repeated quadruple, and the test accounts for this. The vertex n_1 belongs to two planted groups and is therefore very active; a frequency-based method would rank many of its chance co-occurrences highly, while the validation discounts them.

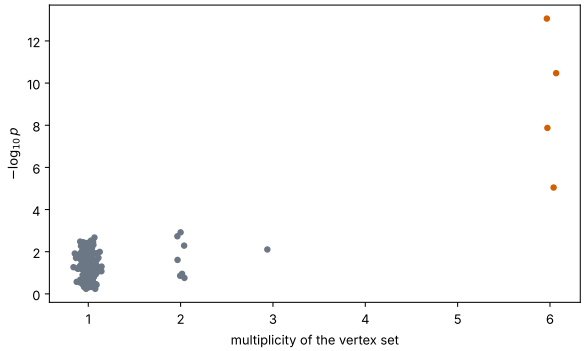


Figure 8.5. Multiplicity and significance ($-\log_{10} p$) of every distinct vertex set of a hypergraph with four planted groups among random hyperedges; the validated sets (orange) are exactly the planted ones.

8.5 Further reading

Agrawal and Srikant [110] and Zaki [111] are the classical references for Apriori and Eclat, Pasquier et al. [112] for closed itemsets, and Ganter and Wille [113] for formal concept analysis. Tan, Kumar, and Srivastava [114] compare the interestingness measures of rules. The association-rule hypergraph of Han et al. [116, 117] is the earliest systematic use of hypergraphs for pattern clustering. Musciotto et al. [118] introduce statistically validated hypergraphs and apply them to social and biological data. The SPMF library [43] implements more than two hundred pattern-mining algorithms whose output `hatvis` reads directly.

CHAPTER

9



Higher-order structure and null models

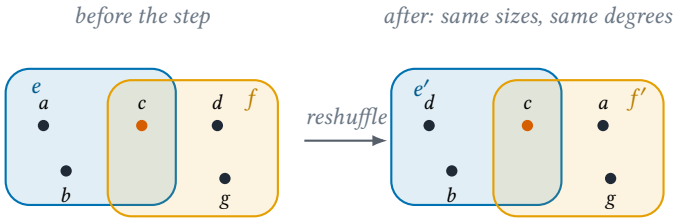
A statistic computed on a single hypergraph means little until it is compared with what chance would produce. This chapter introduces the random hypergraph models of `hatvis`, from the uniform model of HAT to the degree- and size-preserving configuration model of Chodrow, and uses them to test three kinds of structure that have no counterpart in graphs: how three hyperedges overlap (the 26 h-motifs of Lee, Ko, and Shin), how a triple of vertices is covered by pairs and triples (higher-order node motifs), and how close a hypergraph is to a simplicial complex (the simpliciality measures of Landry, Young, and Eikmeier). Clustering coefficients and degree assortativity for hypergraphs complete the toolkit. The examples show that the wire-harness space is far more nested and clustered than its degree and size sequences explain.

9.1 Problem and idea

Suppose the two-node clustering coefficient of a manufacturing space is 0.18. Is that high? A hypergraph in which a few resources take part in most activities and the activities have four or five members will show substantial clustering even if the activities were assembled at random, simply because large hyperedges and hub vertices create many shared neighbors. The meaningful question is whether the observed value exceeds the value expected for hypergraphs *with the same degrees and hyperedge sizes*. The answer requires a null model that preserves exactly these sequences and randomizes everything else, and an ensemble of samples from it.

The same logic applies to every structural pattern of this chapter. An h-motif, a configuration of three overlapping hyperedges, is significant only if it occurs more

often than in the randomized ensemble; a high fraction of hyperedges contained in other hyperedges is interesting only if chance does not produce it. The idea is therefore always the same: compute the statistic, compute it on many samples of a suitable null model, and report the z-score $z = (x_{\text{obs}} - \bar{x}_{\text{null}})/s_{\text{null}}$ or the whole null distribution. The choice of the null model determines what "chance" means, and `hatvis` provides models that preserve progressively more of the observed structure.



The common vertex c stays in both hyperedges; the vertices of the symmetric difference $\{a, b, d, g\}$ are redistributed at random so that $|e'| = |e|$ and $|f'| = |f|$. Every vertex keeps its degree and no hyperedge receives a vertex twice.

Figure 9.1. A pairwise reshuffle step of the hypergraph configuration model (after Chodrow [120]). Repeating the step many times with random pairs of hyperedges samples hypergraphs with the observed degree and size sequences.

9.2 Theory

9.2.1 Random hypergraph models

`hatvis` implements six generators, ordered here by how much of an observed hypergraph they preserve.

Uniform random hypergraph. Each of m hyperedges is a uniformly random k-subset of n vertices. It is the hypergraph analogue of the Erdős–Rényi graph and the generator `uniformErdosRenyi` of HAT [18].

Random hypergraph with a size sequence. Each hyperedge is a uniformly random subset of the prescribed size. It preserves the size distribution but not the degrees, which become approximately binomial; in the comparisons below it is called the *random* model.

Chung–Lu hypergraph. Vertices enter each hyperedge with probability proportional to a target degree, so that the expected degrees match the targets when no vertex dominates [121, 122]. It generates hypergraphs with heterogeneous degrees from scratch, as used in section 6.4.

Configuration model. Chodrow [120] defined configuration models of hypergraphs that preserve the degree of every vertex and the size of every hyperedge,

and sampled them by Markov chain Monte Carlo. The pairwise reshuffle of fig. 9.1 is a simple move of this kind: it picks two hyperedges e, f, keeps their intersection in both, and redistributes the vertices of the symmetric difference at random while keeping both sizes. Every degree and every size is preserved, no vertex appears twice in a hyperedge, and after a number of steps proportional to the number of hyperedges (ten times by default) the sample has forgotten the correlations of the original beyond its degree and size sequences. This is the standard null model for testing higher-order structure.

Planted partition. The hypergraph stochastic block model of section 7.4 draws a hyperedge entirely from one random community with probability p_{in} and from the whole vertex set otherwise; it provides ground truth for community detection. Chodrow, Veldt, and Benson [98] discuss degree-corrected versions and their relation to modularity.

Temporal random hypergraph. A sequence of snapshots in which a tunable fraction of the hyperedges is repeated from the previous step; it serves as a null model for the persistence and turnover measures of chapter 12.

9.2.2 Clustering and assortativity

The clustering coefficient of a graph measures how often two neighbors of a vertex are themselves neighbors. On the clique expansion it is inflated by every hyperedge of size three or more, which contributes complete triangles regardless of any further structure. Gallagher and Goldberg [123] proposed a *two-node clustering coefficient* that asks instead how much the hyperedge memberships of a vertex and its neighbors overlap. With $M(u)$ the set of hyperedges containing u and $N(u)$ its neighbors in the clique expansion,

$$c(u) = \frac{1}{|N(u)|} \sum_{v \in N(u)} \frac{|M(u) \cap M(v)|}{|M(u) \cup M(v)|}, \tag{9.1}$$

the mean Jaccard similarity of the memberships of u and each of its neighbors. It is large when a vertex meets its neighbors repeatedly, in many of the same hyperedges, and small when every neighbor is met once.

The *degree assortativity* of a hypergraph [120] is the Pearson correlation of the degrees of pairs of vertices that share a hyperedge, where the pairs are either all pairs of members or one pair per hyperedge chosen by a rule (the members of largest and smallest degree, for example). Negative values mean that high-degree vertices tend to share hyperedges with low-degree ones, as in a hub-and-spoke organization.

9.2.3 Hypergraph motifs

Network motifs are small subgraphs that occur significantly more often than in randomized networks [124]. Lee, Ko, and Shin [125] defined their hypergraph counterpart for three connected hyperedges. Three hyperedges A, B, C partition the vertices of $A \cup B \cup C$ into seven *regions*: the vertices only in A, only in B, only in C, only in A and B, only in B and C, only in A and C, and in all three. The emptiness pattern of the seven regions, up to permutations of A, B, C, is the *h-motif* of the triple. Excluding patterns with duplicate hyperedges and patterns in which the three hyperedges are not connected leaves 26 h-motifs; 20 of them are *closed* (all three pairs intersect) and 6 are *open* (one pair is disjoint) [125]. The definition is independent of the hyperedge sizes, so the motif profile of hypergraphs with very different sizes can be compared. `hatvis` numbers the motifs $h_1, \ldots, h_{26}$ in the lexicographic order of their canonical patterns, which differs from the figure order of the original paper, and lists the patterns with `h_motif_catalog()`.

Counting h-motifs requires the connected triples of distinct hyperedges, which `hatvis` enumerates from the line graph. The significance of motif k is the z-score of its count against a null ensemble, usually the configuration model, and the vector of z-scores normalized to unit length is the *significance profile* of Milo et al. [126], which makes hypergraphs of different sizes comparable. Lee et al. showed that the h-motif profiles of real hypergraphs are similar within a domain (co-authorship, contact, e-mail, tags) and different across domains.

The complementary view of Lotito et al. [127] looks at small sets of *vertices* and records which of their subsets are hyperedges. For three vertices $\{u, v, w\}$, the hyperedges contained in the triple are a subset of the three pairs and the triple itself. `hatvis` counts the six connected classes: the open wedge (two pairs), the triangle (three pairs), the lone 3-hyperedge, the 3-hyperedge with one or with two of its pairs, and the full simplex (the 3-hyperedge with all three pairs). The last class is the signature of a downward-closed, simplicial organization; the lone 3-hyperedge is the signature of a genuinely higher-order interaction that does not decompose into pairs.

9.2.4 Simpliciality and nestedness

A *simplicial complex* is a hypergraph closed under taking subsets: if $\{a, b, c\}$ is a hyperedge, so are $\{a, b\}$, $\{a, c\}$, $\{b, c\}$, and the singletons. Many models of higher-order dynamics are formulated on simplicial complexes, and the choice between a hypergraph and a simplicial complex as the representation of data should be guided by how much downward closure the data actually show [13, 14]. Landry, Young, and Eikmeier [128] defined three measures, computed on the hyperedges of size at least two (singletons are ignored by default):

- the *simplicial fraction*: the fraction of hyperedges (of size at least three) all of whose subsets of size at least two are hyperedges;

In a nutshell **Higher-order structure and null models**

- Null models: uniform (HAT), size-preserving random, Chung–Lu, configuration model by pairwise reshuffles (degrees and sizes preserved), planted partition, temporal with persistence. Report z-scores or null distributions.
- Two-node clustering (9.1): mean Jaccard overlap of the memberships of a vertex and its neighbors; degree assortativity of co-members.
- h-motifs: emptiness patterns of the seven regions of three connected hyperedges; 26 motifs (20 closed, 6 open); z-scores and significance profile against the configuration model.
- Node motifs of order three: open wedge, triangle, lone 3-hyperedge, partial and full simplex.
- Simpliciality (simplicial fraction, edit and face edit simpliciality) and the included fraction measure the distance from a simplicial complex.

- the *edit simpliciality*: $|E|/|\bar{E}|$, where $\bar{E}$ is the smallest simplicial complex containing E, i.e. the fraction of the simplicial closure that is present;
- the *face edit simpliciality*: the mean, over the maximal hyperedges, of the fraction of their subsets that are present.

All three are 1 for a simplicial complex and close to 0 for a hypergraph without inclusions. They found that many empirical hypergraphs are far from simplicial but much more simplicial than their randomized counterparts. A related, simpler statistic is the *included fraction*: the fraction of hyperedges that are proper subsets of another hyperedge. LaRock and Lambiotte [129] studied this encapsulation structure and its effect on dynamics.

9.3 Using it in `hatvis`

```
motifs.motif_zscores(H, counter, n_samples=30,
null="configuration", random_state=None)
```

`counter` a function that maps a hypergraph to a `Series` of counts, e.g. `motifs.h_motifs` or `motifs.node_motifs`.

Returns observed counts, null mean and standard deviation, z-scores, and the significance profile.

Related `motifs.h_motif_catalog()`, `motifs.h_motif_pattern(a, b, c)`; `models.uniform_random`, `models.random_hypergraph`, `models.chung_lu`, `models.configuration_model(H, n_steps)`, `models.planted_partition`, `models.temporal_random`; `metrics.two_node_clustering`, `metrics.clique_clustering`,

```
    metrics.degree_assortativity,          simpliciality(H),
    metrics.nestedness(H),           metrics.degree_entropy,
    metrics.spectral_entropy; plot_motif_profile.
```

9.4 Examples

h-motifs of the manufacturing space. The h-motifs of the wire-harness space are counted and compared with thirty samples of the configuration model.

```
cat = hv.motifs.h_motif_catalog()
z = hv.motifs.motif_zscores(H, hv.motifs.h_motifs, n_samples=30,
    random_state=0)
```

Four motifs are over-represented (fig. 9.2). The strongest is h_9 (96 instances against 21.3 expected, $z = 7.0$): two of the three activities have no private resources, one of them is contained in the other, and all three share a common core, the typical relation between a short handling step, the station activity that contains it, and a neighboring activity that uses the same sensor or competence. The open motif h_8 (96 against 39.2, $z = 3.1$) is the same nesting without the common core: a contained activity, its container, and a third activity that overlaps the container but not the contained one. The rare motifs h_2 and h_4 (4 instances each, against fewer than one expected) are configurations in which no activity has a private resource at all. The most under-represented motif is h_{23}, the open chain of three activities that each have private resources (2524 instances against 12 706 expected, $z = -17.7$), followed by h_{25} and h_{19}, closed triples in which all three pairs overlap but no resource is shared by all three, and in which at least two activities have private resources. Together the profile says that the activities of the manufacturing space overlap in a nested way, with shared cores and contained steps, while loose chains of activities with their own resources, the generic configuration of randomly assembled groups, are rare.

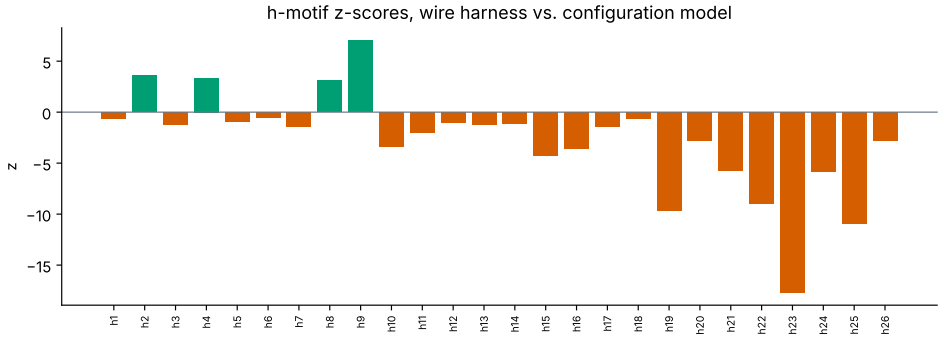


Figure 9.2. z-scores of the 26 h-motifs of the wire-harness space against thirty samples of the configuration model (motifs in the numbering of `hatvis`).

Hypergraph or simplicial complex? Four hypergraphs are compared: a simplicial complex (the downward closure of a random 4-uniform hypergraph), a random hypergraph with sizes 2 to 4, the hypergraph of frequent itemsets of the news data (downward closed by construction, since every subset of a frequent itemset is frequent), and the news articles themselves.

```
table = pd.DataFrame({name: pd.concat([hv.simpliciality(H, max_size=12),
    hv.metrics.nestedness(H)])
                      for name, H in cases.items()}).T
```

Table 9.1. Simpliciality measures, the fraction of hyperedges included in another one, and the counts of the connected node motifs of order three for four hypergraphs.

	simplicial complex	random	frequent itemsets	news articles
simplicial fraction	1.000	0.000	1.000	0.054
edit simpliciality	1.000	0.219	1.000	0.052
face edit simpliciality	1.000	0.018	1.000	0.162
included fraction	0.906	0.111	0.875	0.829
open wedge	66	56	0	0
triangle of pairs	10	0	1	2
lone 3-hyperedge	0	30	0	3
3-hyperedge + 1 pair	0	0	0	29
3-hyperedge + 2 pairs	0	0	0	6
full simplex	31	0	23	13

Table 9.1 and fig. 9.3 confirm the extreme cases: the simplicial complex and the itemset hypergraph score 1 on all three measures, the random hypergraph close to 0. The news articles are instructive. Their simplicial fraction and edit simpliciality are only about 0.05, yet 83% of the articles are subsets of other articles. The articles are highly nested, because many short articles repeat part of the theme set of longer ones, but they are not downward closed, because most subsets of a long article never occur as articles of their own. The node motifs show the same: the dominant class is the 3-hyperedge with only one of its pairs present (29 triples), whereas the simplicial complex consists of open wedges, triangles, and full simplices. Representing the news data as a simplicial complex would add 95% of the faces that do not exist; representing them as a graph would destroy the 3-hyperedges. The hypergraph is the representation that matches the data.

What is structure and what is chance? The last script compares four statistics of the wire-harness space with thirty samples of the configuration model and of the size-preserving random model.

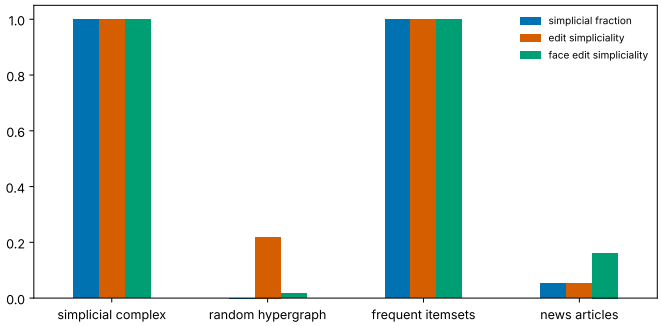


Figure 9.3. Simplicial fraction, edit simpliciality, and face edit simpliciality of a simplicial complex, a random hypergraph, the frequent-itemset hypergraph, and the news articles.

```
    lab = hv.hypergraph_louvain(G, random_state=0)
    return {"two-node clustering":
    hv.metrics.two_node_clustering(G).mean(),
            "degree assortativity": hv.metrics.degree_assortativity(G),
            "modularity": hv.modularity(G, lab),
            "2-components": len(hv.s_connected_components(G, 2))}
```

Table 9.2. Four statistics of the wire-harness space against thirty samples of the configuration model (degrees and sizes preserved) and of the random model (sizes preserved): observed value, null mean ± standard deviation.

Statistic	observed	configuration	random
two-node clustering	0.183	0.097 ± 0.004	0.095 ± 0.003
degree assortativity	−0.134	−0.079 ± 0.038	−0.047 ± 0.040
strict modularity (hypergraph Louvain)	0.419	0.205 ± 0.016	0.242 ± 0.020
number of 2-components	4	7.5 ± 2.1	10.0 ± 2.3

The two-node clustering of the real space is almost twice the null value, more than twenty standard deviations above it: resources meet their neighbors again and again, in many of the same activities, which is what a cellular organization means. The modularity of the best partition is twice the null value; a partition of comparable quality cannot be found in randomized spaces with the same degrees and sizes, so the cells of section 7.4 are not an artifact of the degree sequence. The degree assortativity is negative in all three, but more so in the real space (−0.134), reflecting the hub-and-spoke structure in which the sensors and the AGV share activities with low-degree operators and tools; the difference is about 1.5 null standard deviations from the configuration model, so it is only weakly significant. The real space has fewer 2-components (4) than either null model, because its strong overlaps are concentrated in chains of nested activities rather than scattered. Figure 9.4 shows the null distributions.

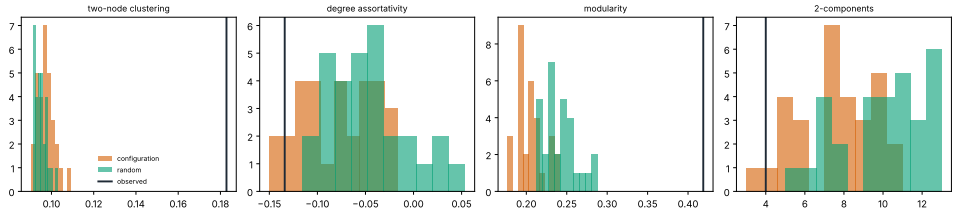


Figure 9.4. Null distributions of four statistics under the configuration model and the random model (thirty samples each); the vertical line is the observed value of the wire-harness space.

9.5 Further reading

Chodrow [120] introduces configuration models of hypergraphs and shows how the choice between vertex-labeled and stub-labeled ensembles affects the conclusions. Lee, Ko, and Shin [125] define h-motifs and give exact and approximate counting algorithms; Lotito et al. [127] study higher-order motifs of vertices and their significance. Landry, Young, and Eikmeier [128] define the simpliciality measures and survey empirical data sets, and LaRock and Lambiotte [129] study encapsulation. Do et al. [85] describe structural patterns and generators of real hypergraphs, and Lee et al. [29] survey the field.

CHAPTER

10



Combinatorial problem solving

Many planning and design questions are classical problems on hypergraphs in disguise: which resources must be monitored so that every activity is observed (a hitting set), which activities cover all resources (a set cover), which groups can run in parallel (a matching), which tables can be joined efficiently (acyclicity and join trees), which products can be made from the available parts and at what cost (reachability and shortest paths in a directed hypergraph), and which engineering specifications should be developed to satisfy the customers (quality function deployment). This chapter states these problems, their integer-programming formulations, the greedy and exact algorithms of `hatvis`, and the structural results (duality, Berge's transversal theorem, the join-tree characterization) that make them tractable or explain why they are hard. The examples come from the manufacturing space, a database schema, a wire-harness assembly, and the electric-car house of quality.

10.1 Problem and idea

A graph models a requirement on pairs: an edge must be covered by one of its endpoints, two adjacent vertices must get different colors, a path passes through pairs. Real requirements often involve groups. An activity is observed if *any one* of its resources carries a sensor; an assembly step can start only when *all* its parts are available; a requirement of a product is met by *some combination* of specifications; a database query joins *several* tables that share attributes. Encoding these as graphs either loses the group (a set cover becomes a vertex cover of the clique expansion, which is a different and usually larger problem) or needs auxiliary constructions that hide the structure.

The idea of this chapter is to formulate the problems directly on the hypergraph, where they have short integer programs and well-understood theory. Two dualities run through the chapter (fig. 10.1). The hitting set of H is the set cover of its dual H^*, so every covering algorithm solves both; and for a simple hypergraph the family of minimal transversals determines the hypergraph again, $Tr(Tr(H)) = H$ [16], so the "what must I monitor" and "what can fail" questions are two views of one object.

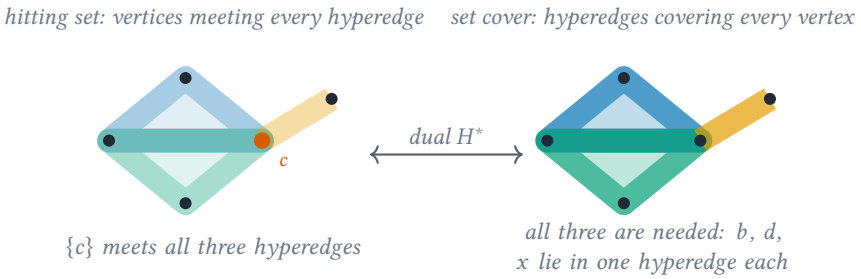


Figure 10.1. Hitting set and set cover are dual problems. Left: a single vertex c meets every hyperedge, so the minimum hitting set has size one. Right: covering all vertices by hyperedges needs all three. A hitting set of H is a set cover of the dual hypergraph H^*, whose hyperedges are the vertex memberships.

10.2 Theory

10.2.1 Covering and hitting

A *set cover* of H is a family of hyperedges whose union contains every vertex (of a given universe U, by default all non-isolated vertices); a *hitting set* or *transversal* is a set of vertices that meets every hyperedge. With costs c_e and c_i, the minimum-cost versions are the integer programs

$$\text{set cover:}\quad \min \sum_e c_e x_e \quad \text{s.t.} \quad \sum_{e \ni i} x_e \geq 1 \;\; \forall i \in U, \quad x_e \in \{0,1\}, \tag{10.1}$$

$$\text{hitting set:}\quad \min \sum_i c_i y_i \quad \text{s.t.} \quad \sum_{i \in e} y_i \geq 1 \;\; \forall e \in E, \quad y_i \in \{0,1\}. \tag{10.2}$$

The constraint matrix of one is the transpose of the other, which is the duality of fig. 10.1. Both are NP-hard [107], but integer programs of a few thousand variables are solved to optimality in seconds by branch and bound; `hatvis` passes them to the HiGHS solver [28] through `scipy.optimize.milp`. The greedy algorithm repeatedly takes the hyperedge with the smallest cost per newly covered vertex, $c_e/|e \cap U|$, until everything is covered; Chvátal [130] proved that its cost is at most $H(\max_e |e|) \leq 1 + \ln \max_e |e|$ times the optimum, where $H(k) = \sum_{j \leq k} 1/j$. The greedy hitting set is the greedy set cover of the dual.

Minimal transversals. A transversal is *minimal* if no proper subset is a transversal. The family of all minimal transversals is the *transversal* (or blocker) hypergraph $Tr(H)$, and for a simple hypergraph $Tr(Tr(H)) = H$ [16]. Minimal transversals answer "what are all the irredundant ways to monitor the system" and, read the other way, "what are all the minimal sets of failures that disable every activity". Berge's sequential algorithm processes the hyperedges one at a time,

$$Tr(e_1, \dots, e_k) = \min \left\{ T \cup \{v\} \;:\; T \in Tr(e_1, \dots, e_{k-1}),\ v \in e_k \right\}, \tag{10.3}$$

where min keeps the inclusion-minimal sets. The output can be exponential in the input, so the right measure of efficiency is the time per output set; Eiter and Gottlob [131] showed that enumerating minimal transversals is equivalent to a family of problems in logic, databases, and artificial intelligence, and Fredman and Khachiyan [132] gave an algorithm that is quasi-polynomial in the combined size of input and output. `hatvis` implements Berge's method with an optional size limit, adequate for subsystems of moderate size.

10.2.2 Matching and coloring

A *matching* is a set of pairwise disjoint hyperedges; a maximum-weight matching, $\max \sum_e w_e x_e$ subject to $\sum_{e \ni i} x_e \le 1$, selects the largest set of groups that can act simultaneously when every vertex can take part in only one group, such as a schedule of activities that do not share resources. A *weak* (proper) coloring assigns colors to vertices so that no hyperedge of size at least two is monochromatic; a *strong* coloring requires all members of every hyperedge to have different colors, which is a proper coloring of the clique expansion. A hypergraph with a weak 2-coloring has *Property B*; deciding Property B is the NP-complete SET SPLITTING problem [133]. Strong colorings assign time slots to the vertices so that no group has two members in the same slot; weak 2-colorings split a set of items into two parts so that every group is represented in both. `hatvis` provides greedy colorings of both kinds and a backtracking decision procedure for Property B on small hypergraphs.

10.2.3 Acyclicity and join trees

A hypergraph is *α-acyclic* if it can be reduced to nothing (or to a single hyperedge) by the *GYO reduction* [134]: repeatedly delete a vertex that belongs to exactly one hyperedge (an *ear vertex*), and delete a hyperedge that is empty or contained in another hyperedge. The order of the deletions does not affect the outcome, and Tarjan and Yannakakis [135] gave a linear-time test. Acyclicity has an equivalent characterization by *join trees*: a tree on the hyperedges is a join tree if, for every vertex, the hyperedges that contain it form a connected subtree (the running intersection property). A hypergraph has a join tree if and only if it is α-acyclic, and the join trees are exactly the maximum-weight spanning trees of the intersection

graph of the hyperedges with weights $|e \cap f|$ [136, 137]. `hatvis` computes a join tree in this way and rejects cyclic inputs.

Acyclicity was identified in database theory as the property of a relational schema that makes queries efficient [138]: the tables are the hyperedges, the attributes the vertices, and for an acyclic schema a join of all tables can be computed by semijoins along the join tree in time polynomial in the size of input and output [139], while for cyclic schemas the intermediate results can blow up. Fagin [140] defined a hierarchy of stronger notions (β-, γ-, and Berge-acyclicity). The same structure appears in probabilistic inference, where junction trees of graphical models are join trees of the hypergraph of cliques, and in constraint satisfaction.

Two further properties are tested by `hatvis`: *conformality* (section 2.2.4) and the *Helly property*, which holds when every pairwise intersecting family of hyperedges has a common vertex. A hypergraph is conformal exactly when its dual has the Helly property, and the Helly property can be tested with the triangle criterion of Berge and Duchet in $O(n^3 m)$ time [16].

10.2.4 Directed hypergraphs: reachability and hyperpaths

A directed hypergraph whose hyperarcs have a single head vertex is a set of *B-arcs*, and it models AND-dependencies: the head can be produced only when every tail vertex is available [33, 34]. A vertex is *B-reachable* from a set S of sources if it is in S or is the head of a hyperarc all of whose tail vertices are B-reachable. The *B-visit* computes the reachable set in linear time by keeping, for every hyperarc, the number of tail vertices not yet reached and firing the hyperarc when the counter drops to zero. A *B-hyperpath* to a vertex is the set of hyperarcs used to reach it.

With a weight w_e on every hyperarc, Gallo et al. [33] define the cost of reaching a head vertex v through hyperarc e as

$$d(v) = \min_{e \,:\, v \in H(e)} \Big(w_e + F\big(\{d(t) \,:\, t \in T(e)\}\big) \Big), \qquad d(s) = 0 \;\; (s \in S), \tag{10.4}$$

with $F = \sum$ (the total effort of all steps needed, counting every sub-assembly) or $F = \max$ (the completion time when independent preparations run in parallel, the makespan). Both are computed by a Dijkstra-like procedure (the SBT procedure of Gallo et al.) that settles the vertices in order of increasing d and evaluates a hyperarc once all its tail vertices are settled. The additive cost counts a sub-assembly once for every use, so it is exact for tree-shaped assemblies and an upper bound when a part feeds several branches. Directed hypergraphs of this kind describe assembly and disassembly sequences, chemical reaction networks [141], Horn-clause inference, and the association rules of section 8.2.2.

10.2.5 Quality function deployment

The house of quality of section 5.2.6 relates customer requirements i with importances λ_i to design parameters (engineering specifications) k by relationship scores

$a_{ik} \in \{0, 1, 3, 9\}$, and its roof records correlations c_{kl} between parameters [80, 82]. The classical evaluation computes the *technical importance*

$$t_k = \sum_i \lambda_i \, a_{ik} \tag{10.5}$$

and develops the parameters in decreasing order of t_k. Abonyi and Czvetkó [3] pointed out that this ranking ignores costs and the fact that parameters act in combination, and proposed three complementary formulations.

Mixed-integer selection. With a binary decision u_k for every parameter, the roof spreads the effect of a developed parameter to its correlated ones, $\tilde{u}_l = \sum_k c_{lk} u_k$ (with $c_{kk} = 1$), and the satisfaction of requirement i is $y_i = \sum_l a_{il} \tilde{u}_l$. The selection maximizes the total satisfaction under a budget C and a limit K on the number of parameters,

$$\max_{u \in \{0,1\}^m} \sum_i y_i \quad \text{s.t.} \quad \sum_k c_k u_k \le C, \quad \sum_k u_k \le K, \quad y_i \ge \lambda_i \text{ (optional)}, \tag{10.6}$$

or minimizes the cost subject to minimum satisfaction levels.

Minimum-cost assignment. A flow network source → parameters → requirements → sink, with capacity 1 and cost c_k on the arc into parameter k, an arc from k to i whenever $a_{ik} > 0$, and demand d_i (default 1) at requirement i, selects the cheapest set of parameters that serves every requirement, each parameter serving one requirement. The model guarantees coverage but treats requirement and parameter in pairs: it cannot reward a parameter that serves several requirements at once.

The hypergraph view. With the requirements as vertices and the parameters as hyperedges (eq. (5.6)), the parameters can be ranked by a hyperedge centrality such as the node–edge eigenvector centrality of section 6.2.5, which rewards a parameter for serving important requirements that are also served by other important parameters, i.e. for its role in the group structure of the house of quality. Abonyi and Czvetkó also defined a *generalized hypergraph* whose hyperedges may contain other hyperedges: the hyperedge of parameter k contains the requirements it influences and the parameters it is correlated with, so that direct and indirect effects can be analyzed in one incidence matrix of size $(|V| + |E|) \times |E|$.

In a nutshell **Combinatorial problems on hypergraphs**

- Set cover (10.1) and hitting set (10.2) are dual (hitting set of H = set cover of H^*); exact by integer programming (HiGHS), greedy within $1 + \ln \max |e|$ of the optimum.
- Minimal transversals by Berge's algorithm (10.3); $Tr(Tr(H)) = H$ for simple H; output-sensitive complexity.
- Matching (disjoint groups), weak and strong coloring, Property B (NP-complete).
- α-acyclicity by GYO reduction; join tree = maximum spanning tree of the intersection graph iff acyclic; efficient joins; conformality and Helly property.
- Directed hypergraphs: B-reachability (linear), minimum hyperpaths with sum or max cost (10.4).
- QFD: technical importance (10.5), MILP selection (10.6), min-cost assignment, hypergraph centrality, generalized hypergraph with correlations.

10.3 Using it in `hatvis`

`problems.hitting_set(H, costs=None)`

Returns a minimum-cost hitting set (list of vertices) by integer programming.

Related `greedy_hitting_set`, `set_cover(H, universe, costs)`, `greedy_set_cover`, `minimal_transversals(H, max_size)`, `transversal_hypergraph`, `maximum_matching(H, weighted)`, `greedy_coloring(H, strong)`, `is_two_colorable`, `gyo_reduction`, `is_alpha_acyclic`, `join_tree`, `is_conformal`, `has_helly_property`, `b_reachable(D, sources)`, `shortest_hyperpaths(D, sources, cost="sum"|"max")`.

`qfd.qfd_select(relationship, costs, correlation=None, requirement=None, budget=None, max_actions=None, objective="max_satisfaction")`

Returns a table with the decision, cost, and contribution of every parameter; `.attrs` holds the total satisfaction, the total cost, and the satisfaction vector.

Related `qfd_min_cost_assignment(relationship, costs, demand)`, `technical_importance`, `qfd_hypergraph`,

> `correlation_hypergraph`, `generalized_qfd_hypergraph`, `plot_house_of_quality`.

10.4 Examples

Minimal monitoring sets. The physical resources of the wire-harness space (operators, robots, the AGV, and the crimping machines) are restricted to the activities in which they take part, and two questions are asked: which resources must carry a sensor so that every activity involves a monitored resource (a hitting set), and which activities must be observed so that every resource is seen (a set cover).

```
hs_exact = problems.hitting_set(people)
hs_greedy = problems.greedy_hitting_set(people)
sc_exact = problems.set_cover(people)
sc_greedy = problems.greedy_set_cover(people)
```

The minimum hitting set has nine resources: six operators (O_2–O_7), the robots R_2 and R_5, and the AGV (fig. 10.2); the greedy rule also finds nine. Every member of this set is forced: each of them is the only physical resource of at least one activity (the AGV of the six transport steps, each operator of a manual handling or assembly step, and R_2 and R_5 of the handling steps into the stations they operate), so any hitting set must contain them, and together they already meet every activity. The operator O_1 is not needed, because every activity of O_1 also involves the AGV. The minimum set cover needs nine activities, the greedy one ten, within Chvátal's bound. Restricted to the first crimping and assembly stations of line 1, the subsystem has exactly two minimal transversals, $\{M_1, O_3\}$ and $\{O_2, O_3\}$: the assembly operator O_3 must be monitored in any case, and the crimping station can be monitored either at its machine or at its operator. Read as failure sets, these are also the two minimal combinations of failures that would stop every activity of the subsystem.

Acyclic schemas and join trees. A small order-processing schema with four tables is a hypergraph whose hyperedges are the tables and whose vertices are the attributes.

```
schema = hv.Hypergraph({"Orders": ["order", "customer", "date"], "Lines":
    ["order", "item", "qty"],
                        "Items": ["item", "supplier", "price"],
    "Customers": ["customer", "city"]})
cycle = hv.Hypergraph({"R": ["a", "b"], "S": ["b", "c"], "T": ["a", "c"]})
for H in (schema, cycle):
    ok, steps = problems.gyo_reduction(H)
    print(f"alpha-acyclic: {ok}; first steps: {steps[:4]}")
tree = problems.join_tree(schema)
```

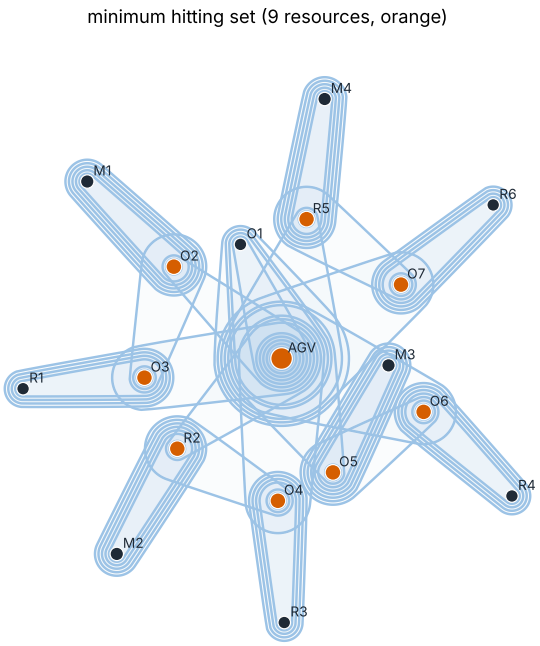


Figure 10.2. A minimum hitting set of the activities of the physical resources of the wire-harness space: the nine resources in orange meet every activity.

The GYO reduction starts by deleting the ear attributes `date`, `qty`, `supplier`, and `price`, each of which occurs in a single table, then removes the tables that have become subsets of others, and continues until nothing is left: the schema is α-acyclic. The join tree links `Customers–Orders` through `customer`, `Orders–Lines` through `order`, and `Lines–Items` through `item` (fig. 10.3); every attribute occurs in a connected part of the tree, so the four tables can be joined by three semijoin passes along it. The three tables $R(a, b)$, $S(b, c)$, $T(a, c)$, in contrast, have no ear vertex and no contained table; the reduction stops immediately, and the 3-cycle is the smallest cyclic schema.

Assembly feasibility and cost. The wire-harness process is reduced to eight steps, each a hyperarc from its input parts to its output: cutting wires, crimping a cut wire with a terminal, mounting a seal in a housing to obtain a connector, inserting the crimped wire into the connector, taping, routing on a jig, and testing; the finished harness can alternatively be bought from a supplier. The weights are the step durations.

```
D = hv.DirectedHypergraph({
    "cut wires":      (["wire"], ["cut wire"]),
    "crimp":          (["cut wire", "terminal"], ["crimped wire"]),
    "mount housing":  (["housing", "seal"], ["connector"]),
    "insert":         (["crimped wire", "connector"], ["sub-harness"]),
```

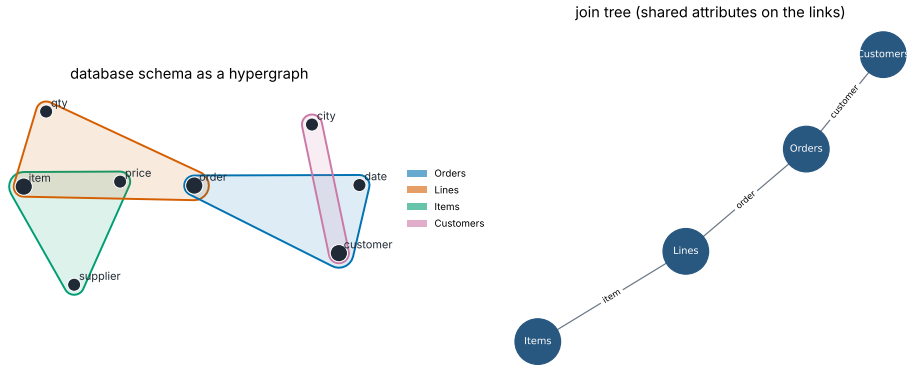


Figure 10.3. A database schema as a hypergraph (left) and its join tree (right), with the shared attributes on the links.

```
    "tape":           (["sub-harness", "tape"], ["taped sub-harness"]),
    "route":          (["taped sub-harness", "jig"], ["harness"]),
    "buy harness":    (["supplier"], ["harness"]),
    "test":           (["harness", "tester"], ["tested harness"]),
}, weights=[2, 3, 3, 4, 2, 5, 30, 3])
available = ["wire", "terminal", "housing", "seal", "tape", "jig",
    "tester"]
print("producible:", sorted(problems.b_reachable(D, available) -
    set(available)))
cost = pd.DataFrame({"sum": problems.shortest_hyperpaths(D, available,
    "sum"),
                     "max": problems.shortest_hyperpaths(D, available,
    "max")})
```

From the available parts all intermediate and final products are B-reachable, but the supplier is not available, so the purchase arc never fires. The minimum hyperpath costs of fig. 10.4 distinguish total effort and makespan. The sub-harness needs 12 time units of work in total (cutting 2, crimping 3, mounting 3, inserting 4), but only 9 if the connector is mounted while the wire is cut and crimped; the tested harness needs 22 units of work and can be completed in 19. The difference is exactly the parallelism of the AND-dependency, which a path in a graph cannot express: in a graph, reaching the sub-harness through the crimped wire would ignore that the connector must also be there.

Which specifications should be developed? The three QFD formulations are applied to the electric-car house of quality of section 5.4 with illustrative development costs (4, 3, 3, 2, 3, 4, 1, 1, 2, 1, 1 for ES1–ES11); the satisfaction of a selection is measured, for comparability, as the sum of its relationship scores.

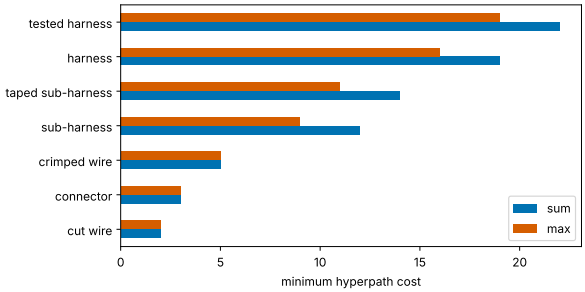


Figure 10.4. Minimum B-hyperpath costs of the products of a simplified wire-harness assembly: total effort (sum over all steps) and makespan with parallel preparation (max over the inputs).

```
lp = hv.qfd.qfd_select(R, costs=costs, budget=12, max_actions=6)
flow = hv.qfd.qfd_min_cost_assignment(R, costs=costs)
_, edge_c = hv.node_edge_eigenvector(q.hypergraph, "linear")
n_lp = int(lp["selected"].sum())                    # as many as the LP
    selected
hyper = edge_c.sort_values(ascending=False).index[:n_lp]
```

Table 10.1. Three readings of the electric-car house of quality with illustrative costs: the selected specifications, their total cost, and their total relationship score.

Model	selected specifications	cost	score
MILP, budget 12, at most 6	ES4, ES5, ES6, ES7, ES10, ES11	12	101
min-cost assignment	ES1, ES4, ES5, ES7, ES8, ES9, ES10, ES11	15	117
hypergraph centrality (top 6)	ES1, ES5, ES6, ES7, ES10, ES11	14	111

Table 10.1 shows how the models answer different questions. The technical importance (10.5) ranks sensors and algorithms first (103 each), followed by LiDAR (95) and GPS navigation (72), without regard to cost. The MILP spends the budget of 12 on the cheap and broadly useful specifications: the vehicle classification, airbags, and the restraint system cost 1 each, grid charging 2, and sensors and algorithms take the rest; LiDAR does not fit. The min-cost assignment must give every requirement its own specification and therefore buys eight, including the number of seats for boot space and the brakes for the braking requirement, at cost 15; it covers every requirement with a dedicated specification, which the MILP does not guarantee. The hypergraph centrality ranks LiDAR, sensors, and algorithms highest because they jointly serve the two most important requirements, and its top six reach a score of 111 at cost 14, more than the MILP but over its budget. None of the three is "the" answer: the MILP is the right tool when the budget is fixed, the assignment when every requirement must have an owner, and the hypergraph view when the question is which specifications form the core of the design. Figure 10.5 shows the house of quality drawn by `hatvis` with an illustrative roof.

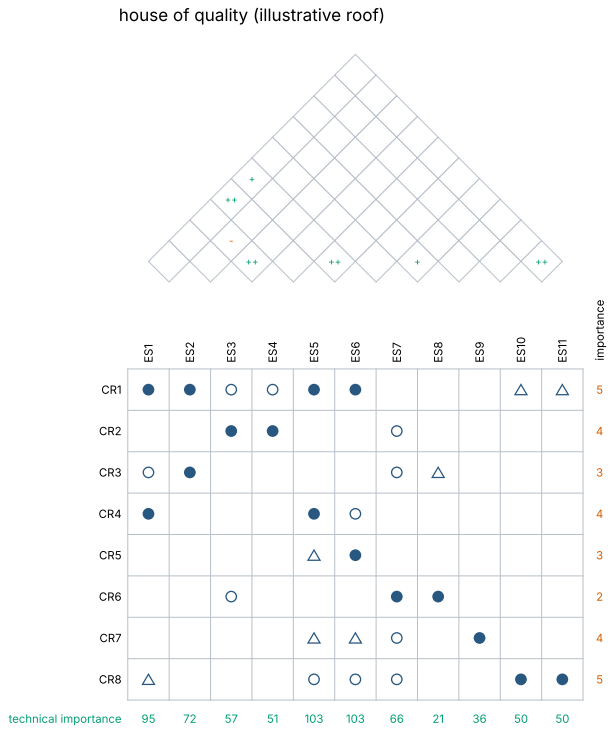


Figure 10.5. The electric-car house of quality drawn by `plot_house_of_quality`: relationship symbols (filled circle 9, circle 3, triangle 1), importance ratings, technical importances, and an illustrative correlation roof.

10.5 Further reading

Berge [16] covers transversals, matchings, colorings, and the Helly property; Chvátal [130] analyzes the greedy set cover, and Eiter and Gottlob [131] and Fredman and Khachiyan [132] the enumeration of minimal transversals. Beeri et al. [138], Fagin [140], and Tarjan and Yannakakis [135] are the classical references on acyclic hypergraphs, and Maier [137] treats join trees in the context of relational databases. Gallo et al. [33] introduce directed hypergraphs, B-reachability, and hyperpaths, and Ausiello and Laura [34] survey the algorithms. The hypergraph and network-flow formulations of QFD are developed in [3]; Chan and Wu [81] review the QFD literature.

CHAPTER

11



Dynamics and learning

A hypergraph is often the substrate of a process: opinions are averaged in groups, infections and innovations spread through teams and households, and labels propagate from a few known vertices to the rest. This chapter shows how the group structure changes such processes. Linear diffusion and consensus can be written with the Laplacians of chapter 2 and are equivalent to processes on a weighted graph; nonlinear group consensus and threshold contagion are not, and produce effects without a pairwise counterpart, such as accelerated or delayed agreement and an abrupt, seed-dependent transition to a spreading state. The second half treats learning on hypergraphs: label spreading, a simplified hypergraph neural network, and the prediction of missing hyperedges.

11.1 Problem and idea

A process on a network is defined by how the state of a node changes as a function of the states of its neighbors. In a pairwise model the influence of a neighbor does not depend on the other neighbors: an infected contact transmits with the same probability whether or not the other members of the household are infected, and the pull of a friend's opinion is the same whether or not the friend agrees with the rest of the group. Many social, biological, and technical processes violate this assumption. People adopt a behavior when a sufficient fraction of a group has adopted it; a group with a coherent opinion is more persuasive than a divided one; a reaction proceeds only when all reactants are present. The interaction is a function of the whole group.

The idea of this chapter is to separate the two cases precisely (fig. 11.1). When the influence of a group is a sum of pairwise influences, the hypergraph process is a graph process on a suitably weighted clique expansion, and the tools of chapter 2 (Laplacians and their spectra) describe it completely. When the influence depends

nonlinearly on the joint state of the group, the process is genuinely higher-order and must be simulated or analyzed on the hypergraph itself [11, 12]. The same distinction applies to learning: a classifier that smooths labels over the clique expansion uses the hypergraph only through pairwise similarities, while edge-dependent weights and nonlinear aggregations use the groups.

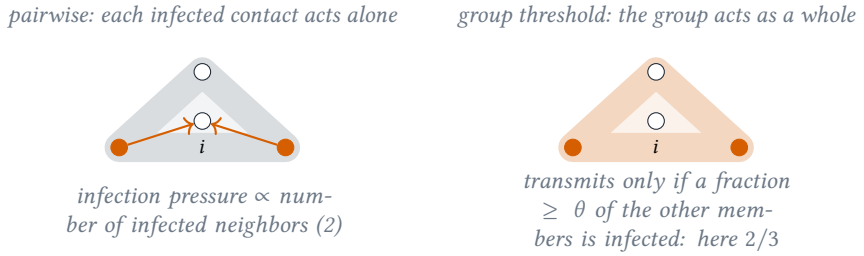


Figure 11.1. Pairwise and group-threshold transmission to a susceptible vertex i in a group of four. In the pairwise model every infected contact transmits independently; in the threshold model the group transmits only when a sufficient fraction θ of the other members is infected. With $\theta = 1$ (simplicial contagion) the group above would not transmit, because one other member is susceptible.

11.2 Theory

11.2.1 Linear diffusion and consensus

Let $x_i(t)$ be the state of vertex i (an opinion, a concentration, a temperature). In linear *averaging dynamics* every vertex moves toward the mean of each of its hyperedges, weighted as in the random walk of (6.4):

$$\dot{\mathbf{x}} = -\mathbf{L}_{\mathrm{rw}}\mathbf{x} = -\left(\mathbf{I} - \mathbf{D}_v^{-1}\mathbf{B}\mathbf{W}\mathbf{D}_e^{-1}\mathbf{B}^{\mathsf{T}}\right)\mathbf{x}. \tag{11.1}$$

The solution is $\mathbf{x}(t) = e^{-t\mathbf{L}_{\mathrm{rw}}}\mathbf{x}(0)$, which `hatvis` evaluates with the action of the matrix exponential on a vector [142] without forming the exponential. On a connected hypergraph all states converge to a common value, the average $\boldsymbol{\pi}^{\mathsf{T}}\mathbf{x}(0)$ weighted by the stationary distribution of the walk ($\pi_i \propto d_i$), and the rate of convergence is the smallest non-zero eigenvalue of $\mathbf{L}_{\mathrm{rw}}$, the algebraic connectivity of section 2.2.5. With Bolla's Laplacian (2.6) the dynamics conserve the total $\sum_i x_i$ instead and describe the diffusion of a quantity. Because $\mathbf{B}\mathbf{W}\mathbf{D}_e^{-1}\mathbf{B}^{\mathsf{T}}$ is the adjacency of a weighted clique expansion (with self-loops), both are graph processes in disguise: the hypergraph enters only through the weights $\sum_{e \ni i,j} w_e/|e|$ of the pairs.

11.2.2 Nonlinear group consensus

Neuhäuser, Mellor, and Lambiotte [143] showed that consensus dynamics become genuinely higher-order when the influence of a group on a member depends on

the agreement of the other members. In their three-body model the pull exerted by a pair of neighbors is modulated by a function of their opinion difference. `hatvis` implements a generalization to hyperedges of any size,

$$\dot{x}_i = \sum_{e \ni i} w_e \, \exp\!\big(\lambda \, \sigma_{e \setminus i}\big) \, \big(\bar{x}_{e \setminus i} - x_i\big), \tag{11.2}$$

where $\bar{x}_{e \setminus i}$ and $\sigma_{e \setminus i}$ are the mean and the standard deviation of the opinions of the *other* members of e. For $\lambda = 0$ the model is linear consensus on the clique expansion with pair weights $w_e/(|e| - 1)$. For $\lambda < 0$ a coherent group (small σ) exerts more influence than a divided one, a form of peer pressure; for $\lambda > 0$ divided groups pull harder. The model is integrated with the explicit Euler method. Neuhäuser et al. showed for the three-body case that such nonlinearities can speed up or slow down the approach to consensus and shift the final consensus value away from the linear average, depending on the initial distribution of opinions and on the structure of the groups.

11.2.3 Threshold contagion

In a discrete-time susceptible–infected–susceptible (SIS) process, every infected vertex recovers with probability μ in each step, and susceptible vertices become infected through their contacts. de Arruda, Petri, and Moreno [144] proposed a threshold rule for hypergraphs: a hyperedge e can transmit to a susceptible member i only if at least a fraction θ of the other members of e are infected, and then it transmits with probability β. With $\theta = 1$ a group transmits only when all other members are infected, the rule of the simplicial contagion model of Iacopini et al. [145]; small θ approaches pairwise spreading. The stationary prevalence ρ^* as a function of β distinguishes the two regimes:

- with pairwise-like transmission, ρ^* rises continuously from zero above an epidemic threshold, and the final state does not depend on the initial fraction of infected vertices;
- with strong group reinforcement, the transition becomes discontinuous and *bistable*: for an intermediate range of β the infection dies out from a small seed but persists from a large one, so the process shows hysteresis.

Iacopini et al. derived this behavior in a mean-field approximation for simplicial complexes, and de Arruda et al. for threshold models on hypergraphs; it means that the outcome of an intervention can depend on the initial condition, not only on the parameters, and that a small increase of transmissibility can cause a sudden jump in prevalence. Mancastroppa et al. [93] showed that such processes localize in the hyper-cores of section 6.2.6.

11.2.4 Semi-supervised learning

When the classes of a few vertices are known, the others can be classified by assuming that vertices in the same hyperedges tend to have the same class. Zhou,

Huang, and Schölkopf [35] extended the label spreading of Zhou et al. [146] to hypergraphs. With the propagation operator $\boldsymbol{\Theta} = \mathbf{D}_v^{-1/2}\mathbf{B}\mathbf{W}\mathbf{D}_e^{-1}\mathbf{B}^{\top}\mathbf{D}_v^{-1/2}$ and the indicator matrix $\mathbf{Y} \in \{0,1\}^{n\times c}$ of the known labels, the class scores

$$\mathbf{F} = (1-\alpha)\,(\mathbf{I} - \alpha\boldsymbol{\Theta})^{-1}\mathbf{Y} \tag{11.3}$$

minimize a smoothness term $\sum_e \frac{w_e}{|e|}\sum_{i,j\in e}\left\|\mathbf{f}_i/\sqrt{d_i} - \mathbf{f}_j/\sqrt{d_j}\right\|^2$ plus a fit term $\frac{1-\alpha}{\alpha}\|\mathbf{F}-\mathbf{Y}\|^2$, and every vertex is assigned to the class of its largest score. The scores of a class grow with the number of its labeled vertices, so an unbalanced labeled set biases the prediction toward the larger class. Zhu, Ghahramani, and Lafferty [147] proposed *class mass normalization* to correct for this: each column of $\mathbf{F}$ is rescaled to a prescribed class mass before the maximum is taken. `hatvis` offers the normalization with uniform class priors as an option.

Hypergraph neural networks [25, 148] stack layers $\mathbf{X}^{(l+1)} = \varphi\big(\boldsymbol{\Theta}\mathbf{X}^{(l)}\boldsymbol{\Psi}^{(l)}\big)$ of propagation with $\boldsymbol{\Theta}$ (with self-loops), a learned linear map $\boldsymbol{\Psi}^{(l)}$, and a nonlinearity φ. Wu et al. [149] showed for graphs that much of the accuracy of such networks is retained when the nonlinearities between layers are dropped: the features are propagated K times, $\boldsymbol{\Theta}^K\mathbf{X}$, and a single linear classifier is trained. `hatvis` implements this simplified HGNN with multinomial logistic regression on standardized propagated features; it needs no deep-learning framework and serves as a strong baseline. Surveys of hypergraph neural networks and representation learning are given by Kim et al. [150] and Antelmi et al. [151].

11.2.5 Hyperedge prediction

Hyperedge prediction asks whether a candidate set of vertices is likely to form a group [152, 153]. `hatvis` evaluates simple, interpretable scores of a candidate S: the mean Jaccard similarity of the members' hyperedge sets, the smallest number of hyperedges shared by a pair of members, and the mean Adamic–Adar index [154] of the member pairs on the clique expansion. To measure predictability, a fraction of the hyperedges is hidden, the scores are computed on the rest, and the hidden hyperedges (positives) are compared with random vertex sets of the same sizes (negatives) by the area under the ROC curve (AUC) and the average precision. Benson et al. [152] showed that groups in temporal data often close over pairs that have already interacted, so that pairwise co-occurrence is a strong predictor; Yadati et al. [153] learn the scores with neural networks.

In a nutshell **Dynamics and learning**

- Linear diffusion and consensus (11.1) are graph processes on a weighted clique expansion; consensus value $\boldsymbol{\pi}^{\mathsf{T}}\mathbf{x}(0)$, rate given by the algebraic connectivity.
- Nonlinear group consensus (11.2): the pull of a group depends on the agreement of its other members; $\lambda < 0$ peer pressure, $\lambda > 0$ the reverse.
- Threshold SIS contagion: a group transmits only if a fraction θ of the other members is infected; $\theta = 1$ is simplicial contagion; discontinuous, bistable transitions with hysteresis.
- Label spreading (11.3) with optional class mass normalization; simplified HGNN ($\boldsymbol{\Theta}^{K}\mathbf{X}$ + logistic regression).
- Hyperedge prediction: Jaccard, co-occurrence, and Adamic–Adar scores; AUC and average precision against size-matched random sets.

11.3 Using it in `hatvis`

`dynamics.threshold_contagion(H, beta, mu, theta=0.5, initial=0.05, n_steps=100, n_runs=1, random_state=None)`

`theta` fraction of the other members that must be infected for a hyperedge to transmit.

`initial` fraction of initially infected vertices or a list of seed vertices.

Returns the prevalence per step for every run (a `DataFrame`).

Related `dynamics.diffusion(H, x0, times, kind)`, `dynamics.group_consensus(H, x0, t_max, dt, lam)`, `dynamics.random_walk_trajectory`; `learning.label_spreading(H, labels, alpha, class_mass_normalization)`, `learning.hgnn_features`, `learning.hgnn_classifier(H, X, labels, n_layers)`, `learning.propagation_operator`, `learning.hyperedge_score`, `learning.hyperedge_prediction(H, test_fraction, method, negative)`.

11.4 Examples

Diffusion and group consensus. A planted-partition hypergraph with 60 vertices in three communities and 120 hyperedges of three or four members ($p_{\text{in}} = 0.85$) carries random initial opinions in $[0, 1]$.

```
lin = hv.dynamics.diffusion(H, x0, np.linspace(0, 15, 61))
runs = {lam: hv.dynamics.group_consensus(H, x0, t_max=3, dt=0.005,
    lam=lam) for lam in (-10.0, 0.0, 10.0)}
```

Under linear diffusion all opinions converge to a common value, the degree-weighted average of the initial opinions (fig. 11.2, left). The right panel compares the decay of the spread of opinions in the nonlinear model (11.2) for three values of λ. With $\lambda = +10$ the spread first falls fastest, because the initially divided groups pull strongly, but the decay then slows down; with $\lambda = -10$ the decay starts slowly, because divided groups have little influence while opinions are diverse, and accelerates as the groups become coherent. At $t = 3$ the spread is 0.0043 for $\lambda = -10$, 0.0051 for the linear model, and 0.0072 for $\lambda = +10$. The same hypergraph and the same initial condition thus give a different time course of agreement depending only on how groups aggregate the opinions of their members, an effect that no reweighting of pairs can reproduce.

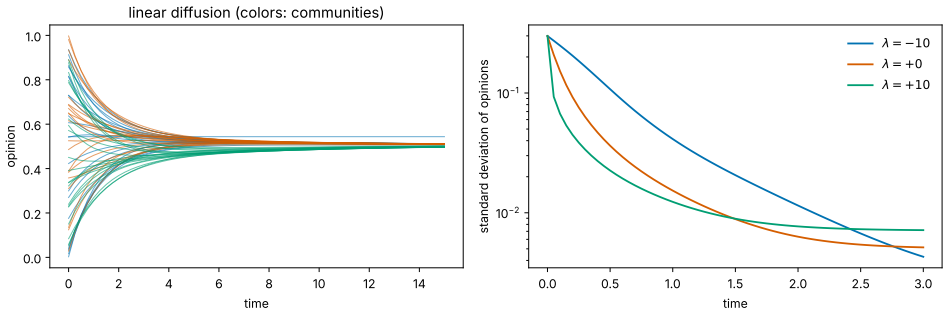


Figure 11.2. Left: linear diffusion of opinions on a planted-partition hypergraph (colors: communities). Right: standard deviation of the opinions over time in the nonlinear group consensus model for $\lambda = -10$, 0, and $+10$.

Threshold contagion. A random hypergraph with 150 vertices, 500 hyperedges of size three, and 150 of size two carries an SIS process with recovery probability $\mu = 0.2$, for a range of transmission probabilities β, two thresholds, and two initial seeds (5% and 40% of the vertices).

```
        run = hv.dynamics.threshold_contagion(H, beta=b, mu=0.2,
theta=theta, initial=init, n_steps=120,
                                              n_runs=3, random_state=0)
```

With $\theta = 0.5$ a group of three transmits as soon as one of the two other members is infected, the process behaves like pairwise spreading, and the stationary prevalence is the same for both seeds at every β (0.43 already at $\beta = 0.02$, rising smoothly to 0.83). With $\theta = 1$ a group of three transmits only when both other members are infected. At $\beta \approx 0.06$ the infection dies out from the 5% seed but reaches a prevalence of 0.42 from the 40% seed; at $\beta \approx 0.10$ the two seeds give 0.48 and 0.71 (fig. 11.3). The region of bistability is the hysteresis predicted for

simplicial and threshold contagion [144, 145]: whether a behavior spreads depends on how many adopt it initially, not only on how contagious it is.

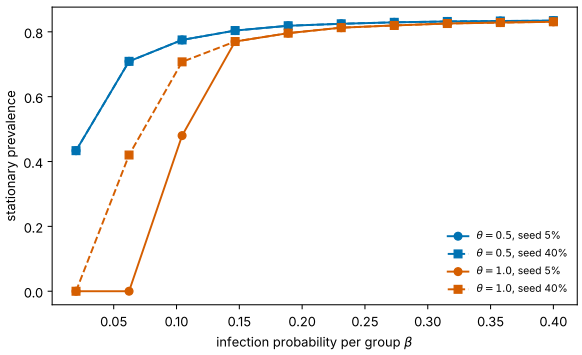


Figure 11.3. Stationary prevalence of threshold SIS contagion as a function of the transmission probability per group, for thresholds $\theta = 0.5$ (blue) and $\theta = 1$ (orange) and initial seeds of 5% (solid) and 40% (dashed).

Learning from few labels. A planted-partition hypergraph with 150 vertices in three classes and 400 hyperedges of two to four members ($p_{\text{in}} = 0.75$) is classified from a random fraction of labeled vertices, five repetitions per fraction, with label spreading, label spreading with class mass normalization, and the simplified HGNN with identity input features.

```
    pred_ls = hv.learning.label_spreading(H, lab,
alpha=0.9).idxmax(axis=1)
    pred_cm = hv.learning.label_spreading(H, lab, alpha=0.9,
class_mass_normalization=True).idxmax(axis=1)
    pred_hg = hv.learning.hgnn_classifier(H, np.eye(H.n_nodes), lab,
n_layers=2).astype(float)
```

Table 11.1. Mean accuracy of semi-supervised classification on a planted-partition hypergraph against the fraction of labeled vertices (five repetitions).

labeled	label spreading	label spreading, class mass norm.	simplified HGNN
2%	0.778	0.929	0.669
5%	0.836	0.927	0.851
10%	0.713	0.965	0.911
20%	0.912	0.971	0.976
40%	0.975	0.989	0.991

Table 11.1 and fig. 11.4 show that plain label spreading is unstable: its accuracy drops to 0.713 at 10% labels. The drop comes from unbalanced labeled sets; in one repetition nine vertices of one class and three of each of the others were labeled, and label spreading assigned 130 of the 150 vertices to the overrepresented class.

Class mass normalization removes the bias and gives the best accuracy at low label rates (0.93 with 2% labels, i.e. three or four labeled vertices). The simplified HGNN needs more labels, because its logistic regression has to be trained on the labeled vertices, but it is the most accurate from 20% on (0.976 and 0.991).

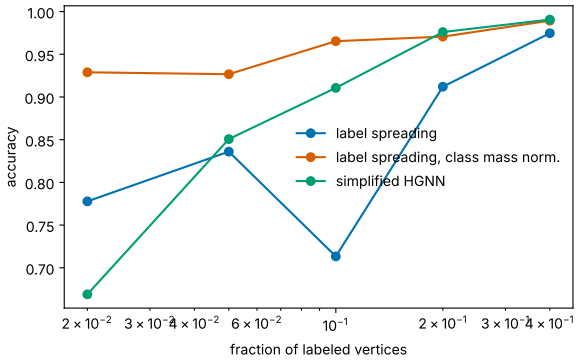


Figure 11.4. Accuracy of label spreading, label spreading with class mass normalization, and the simplified HGNN against the fraction of labeled vertices (logarithmic axis).

Predicting hyperedges. Twenty percent of the hyperedges of the same hypergraph are hidden and scored against size-matched random vertex sets.

```
pred = pd.DataFrame({m: hv.learning.hyperedge_prediction(H, method=m,
    random_state=0)
                     for m in ("mean_jaccard", "min_cooccurrence",
    "adamic_adar")}).T
```

The Adamic–Adar score reaches an AUC of 0.655, the mean Jaccard score 0.593, and the minimum co-occurrence 0.525, barely above chance. The planted partition has a weak signal for hyperedge prediction: a hidden hyperedge is internal to a community with probability 0.75, but a random set of two to four of 150 vertices is often internal to one of three communities as well. Scores based on shared neighbors (Adamic–Adar) capture the community signal best; the minimum co-occurrence is too strict, since in a sparse hypergraph most member pairs of a hidden hyperedge never met elsewhere.

11.5 Further reading

Battiston et al. [11] review dynamics on higher-order networks, including diffusion, synchronization, contagion, and evolutionary games. Neuhäuser et al. [143] analyze nonlinear consensus, Iacopini et al. [145] simplicial contagion, and de Arruda et al. [144] threshold contagion on hypergraphs; Lucas et al. [41] treat synchronization with a multiorder Laplacian. Zhou et al. [35] introduce spectral learning on hypergraphs; Feng et al. [148] and Gao et al. [25] hypergraph neural

networks; Kim et al. [150] and Antelmi et al. [151] survey the field. Benson et al. [152] study simplicial closure and higher-order link prediction.

CHAPTER

12



Temporal and comparative analysis

Hypergraphs change: teams form and dissolve, news narratives rise and fade, and production systems are redesigned. This chapter treats a hypergraph with time stamps as a sequence of snapshots and measures how the groups turn over, which combinations persist, how the activity of vertices shifts, and how frequent patterns differ between periods. The same tools compare two versions of a system: a difference hypergraph marks what was kept, removed, and added, group statistics compare the composition of the work, and distances between hypergraphs summarize the change in one number. The examples follow the narratives of the synthetic news data and the redesign of the production system of HyperGodot.

12.1 Problem and idea

A static hypergraph aggregates all groups regardless of when they occurred. For many questions the timing is the point. A narrative that dominates the coverage of one year and disappears the next, a team that keeps reforming with the same core, or a production step that was removed in a redesign cannot be seen in the aggregate. Temporal properties of group interactions differ systematically from those of pairs: Cencetti et al. [155] found, in face-to-face contact data, that larger groups are shorter-lived and that group composition changes in characteristic ways that pairwise contact networks do not capture.

The idea of this chapter is to keep the time stamp as an attribute of every hyperedge and to derive from it three kinds of objects (fig. 12.1): *snapshots*, the hypergraphs of the hyperedges in one period, to which every static method applies; *transitions* between consecutive snapshots, measured by turnover and persistence;

and *differences* between two hypergraphs, marked hyperedge by hyperedge. The PAOH view of section 3.2.6 and the group-tagged drawings of section 4.2.5 are the visual counterparts.

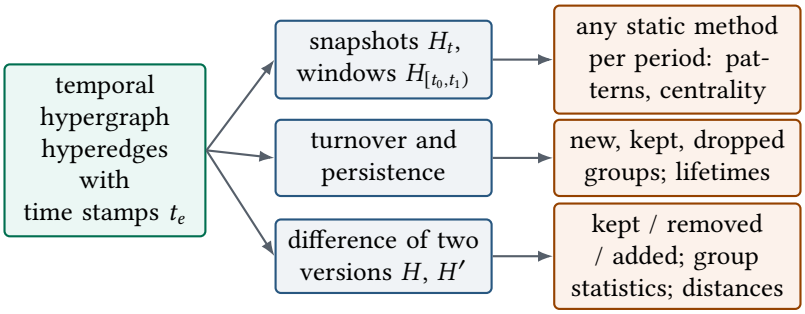


Figure 12.1. Temporal and comparative analysis of hypergraphs. Time stamps on the hyperedges define snapshots and windows to which every static method applies, transitions between snapshots measured by turnover and persistence, and differences between two versions of a system.

12.2 Theory

12.2.1 Snapshots and windows

A *temporal hypergraph* assigns a time stamp t_e (a number, a date, or an ordered label such as a year) to every hyperedge; a repeated group at a different time is a separate hyperedge. The *snapshot* at time t is the partial hypergraph $H_t = (V, \{e : t_e = t\})$ and the *window* $[t_0, t_1)$ is $H_{[t_0,t_1)} = (V, \{e : t_0 \le t_e < t_1\})$; keeping the full vertex set makes the snapshots directly comparable. Any measure of the previous chapters can be computed per snapshot, giving a time series per vertex (degree, centrality) or per pattern (support). The *aggregate* hypergraph merges repeated vertex sets over time and adds their weights.

12.2.2 Turnover and persistence

Let $\mathcal{S}_t$ be the set of distinct vertex sets that occur as hyperedges in snapshot t. Between consecutive snapshots, the *new* groups are $\mathcal{S}_{t+1} \setminus \mathcal{S}_t$, the *persisting* groups $\mathcal{S}_t \cap \mathcal{S}_{t+1}$, and the *dropped* groups $\mathcal{S}_t \setminus \mathcal{S}_{t+1}$; the Jaccard similarity

$$J_t = \frac{|\mathcal{S}_t \cap \mathcal{S}_{t+1}|}{|\mathcal{S}_t \cup \mathcal{S}_{t+1}|} \tag{12.1}$$

summarizes the stability of the group repertoire. For every distinct vertex set, its *persistence* is the number of time stamps in which it occurs, with its first and last occurrence. Groups that persist across all periods are the stable core of a system; groups that occur once are transient. The temporal random model of section 9.2.1,

in which a fraction of the hyperedges of each step is copied from the previous one, provides a null model for these quantities.

12.2.3 Patterns over time

Frequent itemsets mined per snapshot (chapter 8) show which combinations dominate each period; comparing their supports shows which combinations grow or decline. Because the support is relative to the snapshot, a combination can become more frequent in relative terms even when the number of hyperedges changes. For planted or labeled data the share of hyperedges with a given label per period (here, the share of articles that carry a given narrative) is the corresponding ground-truth series.

12.2.4 Comparing two hypergraphs

Two versions of a system, before and after a redesign, are compared hyperedge by hyperedge: the *difference hypergraph* is the union of both, with every hyperedge marked as present in both versions, only in the first, or only in the second, where hyperedges are matched by their vertex sets. Drawn with the status as the group tag (section 4.2.5), it shows at a glance what was kept, removed, and added. Group statistics per version (number, total weight, and mean size of the hyperedges per tag, the vertices involved, the most involved vertex) quantify how the composition of the work changes, for example between human and machine tasks.

A single number for the size of the change is given by a distance between hypergraphs. `hatvis` provides four that look at different levels: the Jaccard distance $1 - |E_1 \cap E_2|/|E_1 \cup E_2|$ of the sets of hyperedges (which hyperedges changed), the normalized Hamming distance of the clique-expansion adjacency patterns on the union of the vertex sets (which pairs changed), the Euclidean distance of the sorted spectra of the normalized Laplacians (how the global connectivity changed), and the ℓ_1 distance of the normalized degree distributions (how the load on the vertices changed). Surana, Chen, and Rajapakse [19] develop a framework of hypergraph similarity measures based on graph and tensor representations, and Felippe, Kirkley, and Battiston [84] an information-theoretic one that corrects for spurious correspondences.

In a nutshell **Temporal and comparative analysis**

- Snapshots and windows are partial hypergraphs on the full vertex set; every static method applies per period.
- Turnover between consecutive snapshots: new, persisting, and dropped vertex sets and their Jaccard similarity (12.1); persistence and lifetime of every group.
- Frequent patterns per period and their changing supports.
- Difference hypergraph (both / only first / only second), group statistics per tag, and distances between hypergraphs (Jaccard on hyperedges, Hamming on pairs, spectral, degree).

12.3 Using it in `hatvis`

`TemporalHypergraph(edges, times, **kwargs)`

Methods `times`, `at(t)`, `window(start, stop)`, `snapshots()`, `aggregate(how)`, `TemporalHypergraph.from_hypergraph(H, time_attr)`.

Related `temporal.turnover(T)`, `temporal.persistence(T)`, `temporal.temporal_centrality(T, func)`, `temporal.difference(H1, H2, labels)`, `temporal.group_statistics(H, group, weight_name)`, `similarity.hypergraph_distance(H1, H2, kind)`; plots `plot_paoh(T, time="time")`, `plot_activity(T)`.

12.4 Examples

Evolving narratives. The synthetic news hypergraph with 150 articles per period (section 3.4) is analyzed period by period.

```
print(hv.temporal.turnover(T))
print(hv.temporal.persistence(T).head(5).to_string())
top = {t: hv.frequent_itemsets(S, min_support=0.08, min_size=2).head(3)
    for t, S in T.snapshots().items()}
```

The repertoire of theme combinations is volatile: from 2021 to 2022, 94 combinations are new, 29 persist, and 86 are dropped (Jaccard similarity 0.139); from 2022 to 2023 the numbers are 90, 26, and 97 (0.122). Most articles use a combination that occurs in one period only, which is expected when articles draw random subsets of their narratives. The persistent combinations are the stable cores: the four-theme sets of the climate-science, impacts, and society narratives occur in all three periods, as do two large combinations that join finance and loss-and-

damage with the impacts or the policy themes. The most frequent pairs of each period trace the planted trends. In 2021 they come from climate science, energy, and policy (EMISSIONS–TEMPERATURE, COAL–OIL_GAS, AGREEMENT–PLEDGE); in 2022 all three come from climate science; in 2023 all three come from the finance narrative (DEVELOPING–LOSS_DAMAGE, AID–LOSS_DAMAGE, DEVELOPING–FINANCE). The ground truth confirms the shift: the share of articles that carry the finance narrative grows from 0.10 to 0.19 and 0.34. The activity plot of fig. 12.2 shows the same at the level of single themes: the finance and loss-and-damage themes grow from period to period while the energy themes decline.

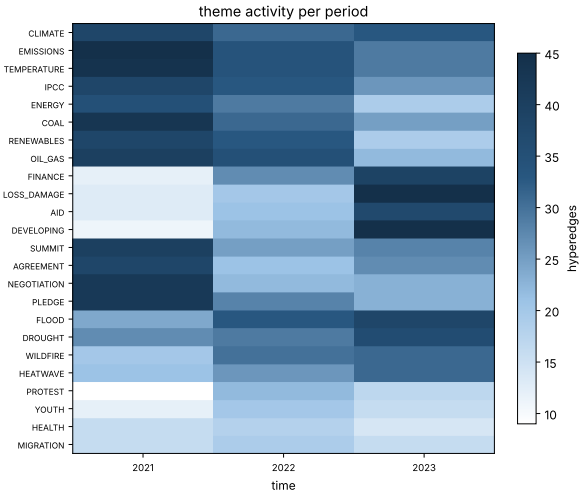


Figure 12.2. Activity (number of articles) of every theme in the three periods of the synthetic news data; themes are ordered by narrative.

Comparing two versions of a production system. The HyperGodot case study [2] suggested two changes to the production example: dropping the final inspection E_7, which duplicates the automated quality control E_6, and automating the manual assembly E_5. The redesigned system replaces E_5 by a robotic assembly E_{5r} performed by a new robot R_3 on the same components in 18 minutes (an illustrative time).

```
after = before.remove_edges(["E7", "E5"]).add_edges({"E5r": ["R3", "C2",
    "C3", "C4", "FP"]}, weights=[18])
after.edge_data.loc["E5r", ["task", "group"]] = ["Robotic assembly", "M"]
diff = hv.temporal.difference(before, after, labels=("before", "after"))
stats = pd.concat({"before": hv.temporal.group_statistics(before,
    weight_name="minutes"),
                   "after": hv.temporal.group_statistics(after,
    weight_name="minutes")})
```

Five tasks are kept, two are removed (E_5 and E_7), and one is added (E_{5r}). The human group shrinks from five tasks with 85 minutes to three tasks with 40 minutes, while the machine group grows from two tasks with 45 minutes to three

with 63 minutes. The difference hypergraph of fig. 12.3 draws the kept tasks solid and the removed and added ones dashed and dotted; the robotic assembly takes the place of the manual one at the junction of the components. The operator W_4 takes part only in the two removed tasks, so the redesign frees this operator entirely.

```
dist = {kind: hv.similarity.hypergraph_distance(before, after, kind) for
    kind in ("jaccard", "hamming", "spectral", "degree")}
```

The four distances describe the redesign at four levels. The Jaccard distance of the hyperedge sets is 0.375 (three of eight distinct tasks changed), the Hamming distance of the pairwise adjacencies only 0.067 (most pairs of entities still work together, because the new task joins the same components), the spectral distance 0.094, and the degree distance 0.258, reflecting that the load moves from the worker to the new robot.

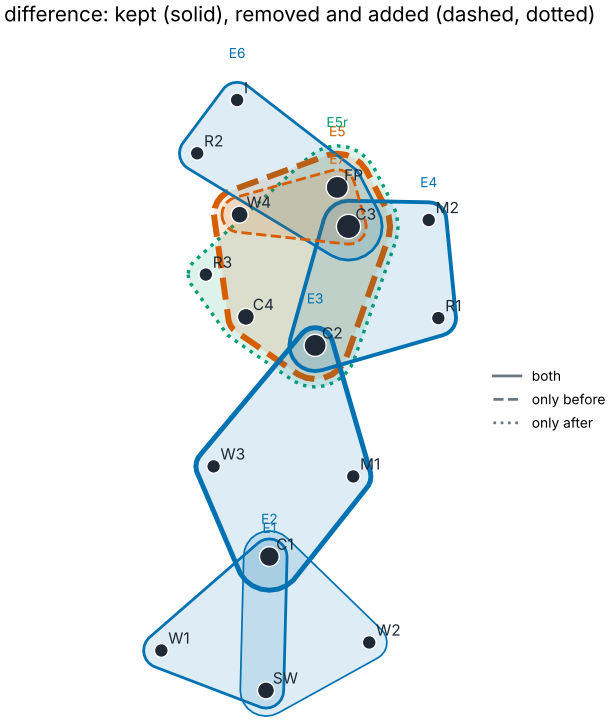


Figure 12.3. Difference hypergraph of the production example before and after the redesign: tasks kept (solid), removed (dashed: manual assembly E_5, final inspection E_7), and added (dotted: robotic assembly E_{5r}).

12.5 Further reading

Valdivia et al. [53] design the PAOH view for dynamic hypergraphs and evaluate it with historians and social scientists. Cencetti et al. [155] analyze the temporal properties of group interactions in social data. Surana, Chen, and Rajapakse [19] and Felippe, Kirkley, and Battiston [84] develop measures for comparing hypergraphs. Fischer et al. [31] survey static and dynamic hypergraph visualizations.

CHAPTER

13



What the hypergraph view buys

> The previous chapters introduced one method at a time. This chapter steps back and collects what the examples have shown about the value of representing group interactions as hypergraphs: where the hypergraph changed a conclusion, where it made a pattern visible, and where it was not needed. It ends with a guide from questions to methods, a workflow for a new data set, and the limits of the toolbox.

13.1 The evidence of the examples

Every chapter contains at least one comparison between a hypergraph method and its pairwise or projected counterpart. Taken together, they support five claims.

The incidence matrix stores what the projections lose. Two hypergraphs that differ only in whether three vertices act together or in pairs have identical clique expansions (section 2.5); every statistic of the expansion is therefore blind to the difference. The conformality test of section 2.2.4 says when the loss is harmless, and the simpliciality measures of section 9.2.4 say how far a data set is from a simplicial complex: for the news data, representing the articles as a simplicial complex would have added about 95% non-existent faces, and representing them as a graph would have destroyed every three-theme combination.

Overlap strength changes the ranking of elements. In the wire-harness space, the crimping competence C_7 ranks eleventh by s-betweenness when any shared activity counts ($s = 1$) and second when only strong overlaps count ($s = 2$) (section 6.4). A pairwise analysis has only the first view; the s-family shows that C_7 is the element through which the strongly overlapping crimping activities

hang together, which is exactly what a redesign of the cells has to respect. The s-component profile showed in the same way that the activities are 1-connected with diameter 2 but split into seventeen components at $s = 3$: the system is a set of tight cells connected by weak links.

Group-aware objectives find group-defined structure. When communities are defined by whole hyperedges being internal, the strict hypergraph modularity recovers them under more noise than modularity on the clique expansion (an adjusted Rand index of 0.956 against 0.872 at $p_{\text{in}} = 0.45$, section 7.4), and on the manufacturing space it finds a partition of higher hypergraph modularity (0.419 against 0.394). The null models of section 9.4 confirm that this modularity, and the two-node clustering of 0.183, are about twice what the degree and size sequences alone produce.

Patterns are hyperedges, and hyperedges can be tested. Frequent itemsets, association rules, and validated groups are all hyperedges of derived hypergraphs. Treating them as such allowed the same drawings, clusterings, and statistics to be applied to them: ARHP clustering of the itemset hypergraph recovered the six planted narratives exactly, including one that the maximal itemsets captured only partially (section 8.4), and the statistical validation of section 8.2.4 singled out exactly the four planted groups among 300 random ones, ranking a repeated quadruple as far more surprising than a repeated pair although both occurred six times. The h-motif profile showed that the activities of the manufacturing space overlap in a nested way, with loose chains of activities strongly under-represented (section 9.4).

Group effects change the dynamics. When a group transmits only if all its other members are infected, the stationary prevalence depends on the initial seed: at the same transmission probability the infection died out from 5% and persisted at 42% from 40% (section 11.4). Nonlinear group consensus changed the time course of agreement on the same hypergraph and from the same start. In a directed hypergraph the cost of an assembly distinguished total effort (22 time units) from makespan (19) because an AND-dependency waits for all its inputs (section 10.4); no path in a graph expresses this.

Where the hypergraph was not needed. The examples also show the limits. On a uniform hypergraph PageRank of the Zhou walk is essentially the degree (Spearman correlation 0.99, section 6.4), and linear diffusion and label spreading are processes on a weighted clique expansion (section 11.2.1). When a question is answered by a pairwise statistic, the projection is the efficient choice; the point of the hypergraph is that the projection is then a deliberate decision with a known loss, not the only option the software offers.

13.2 From questions to methods

Table 13.1 maps typical questions to the representation, the method, and the function of `hatvis` that answers them.

Table 13.1. From questions to methods: the representation, the method, and the `hatvis` function for typical questions about group interactions.

Question	Representation	Method	Function (chapter)
Can the data be stored without loss and exchanged?	incidence matrix, HIF	interchange, converters	`io.write_hif` (2)
Does a projection lose information?	clique expansion	conformality, simpliciality	`problems.is_conformal`, `simpliciality` (2, 9)
Which vertices and hyperedges belong together in the matrix?	incidence matrix	seriation, co-clustering	`seriate`, `seriation.bicluster_blocks` (3)
How can the groups be drawn legibly?	layout + hyperedge style	force, similarity, membership-aware layouts; hulls, isocontours	`draw_hypergraph`, `layout_membership` (4, 5)
Which requirements or resources are similar?	weighted incidence profiles	Jaccard, transitive closure, MDS	`vertex_similarity`, `draw_hatvis` (5)
How strongly connected is the system, and which elements are critical?	s-line graphs	s-components, s-betweenness, averages over s	`s_component_profile`, `s_centrality_table` (6)
Which vertices matter if a group needs all its members?	adjacency tensor, incidence	ZEC, HEC, node–edge centralities, hyper-cores	`z_eigenvector`, `node_edge_eigenvector`, `hypercoreness` (6)
Which communities or cells exist?	hypergraph, projections	strict modularity, Louvain, spectral, co-clustering	`hypergraph_louvain`, `spectral_coclustering` (7)
How to split with little cut?	hypergraph	FM bisection, normalized cut	`partition.fm_bisection` (7)
Which combinations recur, and which predict others?	transactions	Eclat, closed itemsets, rules, ARHP	`frequent_itemsets`, `association_rules` (8)
Is a recurring group significant?	hyperedges per size	iterated hypergeometric test	`patterns.validated_hyperedges` (8)
Is a structure more than chance?	null ensembles	configuration model, motif z-scores	`models.configuration_model`, `motifs.motif_zscores` (9)
What must be monitored or covered?	hypergraph, dual	hitting set, set cover, transversals	`problems.hitting_set` (10)
Can tables be joined efficiently?	schema hypergraph	GYO reduction, join tree	`problems.join_tree` (10)
What can be produced, at what cost?	directed hypergraph	B-reachability, hyperpaths	`problems.shortest_hyperpaths` (10)
Which specifications to develop?	house of quality	MILP, min-cost flow, centrality	`qfd.qfd_select` (10)

Table 13.1, continued

Question	Representation	Method	Function (chapter)
How do group processes behave?	Laplacians, hyperedges	diffusion, group consensus, threshold contagion	`dynamics.*` (11)
How to classify from few labels or predict groups?	propagation operator	label spreading, simplified HGNN, scores	`learning.*` (11)
How does the system change?	snapshots, difference	turnover, persistence, distances	`temporal.*` (12)

13.3 A workflow for a new data set

Figure 13.1 summarizes the order in which the methods of the book are usually applied to a new data set. The first steps decide whether the hypergraph is the right representation at all (are there genuine groups, are they nested or downward closed, are there duplicates and equivalent vertices). The middle steps describe the structure and test it against null models before interpreting it. The last steps answer the substantive question and communicate the result.

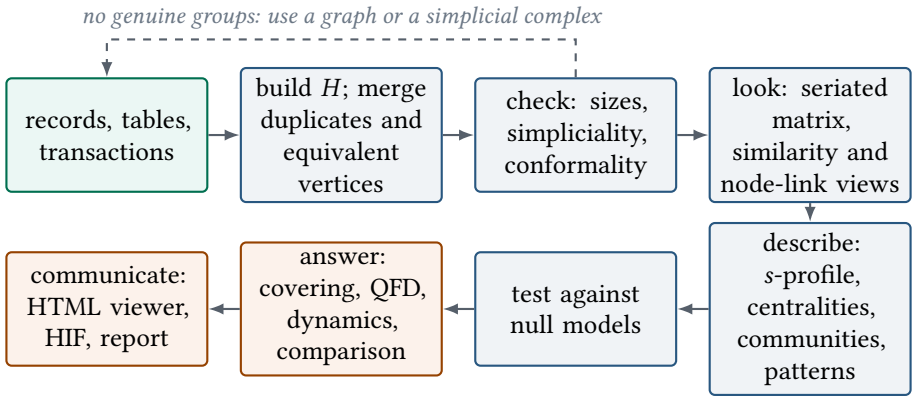


Figure 13.1. A workflow for the hypergraph analysis of a new data set. The dashed branch is a legitimate outcome: if the groups turn out to be pairs or downward-closed families, a graph or a simplicial complex is the better representation.

13.4 Limits and outlook

`hatvis` is designed for hypergraphs that can be inspected: up to a few thousand vertices and hyperedges for the drawings, and up to tens of thousands of incidences for the analyses, most of which work with sparse matrices. Several methods are exponential in the worst case (minimal transversals, Property B, simplicial closure)

or cubic in the number of vertices (the transitive closure of similarities, classical MDS), and the tensor centralities are restricted to uniform hypergraphs. For much larger data the converters and the HIF interchange of section 2.3 hand the data to packages built for scale, and the results come back by label.

Three directions of current research will shape the next versions. Visualization of large hypergraphs relies increasingly on simplification that preserves topological structure [69, 70, 71], which fits the separation of layout and hyperedge style used here. Statistical inference on hypergraphs, from generative community models [98, 104] to the validation of groups [118] and information-theoretic comparison [84], is moving from heuristics to principled models. Learning on hypergraphs [150, 151] is turning the propagation operators of section 11.2.4 into trainable architectures. The interchange format [26] is what makes it possible to combine these developments across packages, and `hatvis` will follow it.

APPENDIX

A



API reference

This appendix lists every public class and function of `hatvis`, 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(hatvis.layout.layout_membership)`). The conventions shared by all functions are described in section 1.5, and the notation of the docstrings is that of table 1.2.

A.1 `hatvis.core`: Hypergraph data structures

Name	Kind	Description
`Hypergraph`	class	Undirected (multi-, weighted, fuzzy) hypergraph
`DirectedHypergraph`	class	Directed hypergraph: every hyperarc $e = (T(e), H(e))$ has a tail and a head set
`TemporalHypergraph`	class	Hypergraph whose hyperedges carry a time stamp (or any ordered group label)

A.2 `hatvis.representations`: Expansions, matrices, and tensors

Name	Kind	Description
`clique_adjacency`	function	Weighted adjacency matrix of the clique expansion (2-section)
`clique_expansion`	function	Clique expansion as a weighted networkx.Graph (edge attribute weight)
`star_expansion`	function	Star expansion: the bipartite vertex–hyperedge (Levi / König) graph
`line_graph`	function	The s-line graph $L_s(H)$ of the hyperedges (Aksoy et al. 2020)
`s_line_graph`	function	s-line graph of the hyperedges (edges=True) or of the vertices
`intersection_matrix`	function	Hyperedge intersection sizes $\lvert e \cap f \rvert = (\mathbf{B}^\top\mathbf{B})_{ef}$ (diagonal: sizes)
`cooccurrence_matrix`	function	Number of shared hyperedges $(\mathbf{B}\mathbf{B}^\top)_{ij}$ (diagonal: degrees)
`laplacian`	function	Hypergraph Laplacian matrices (vertex space, $n \times n$)
`adjacency_tensor`	function	Adjacency tensor of a k-uniform hypergraph (Cooper and Dutle 2012)
`degree_tensor`	function	Diagonal degree tensor $\mathcal{D}$ (d_i on the super-diagonal) of a uniform hypergraph
`laplacian_tensor`	function	Laplacian tensor $\mathcal{L} = \mathcal{D} - \mathcal{A}$ of a uniform hypergraph (as in HAT)
`tensor_apply`	function	Tensor–vector product $(\mathcal{A}\mathbf{x}^{k-1})_i = \sum_{e \ni i} w_e \prod_{j \in e \setminus i} x_j$
`uniform_layers`	function	Split a hypergraph into its uniform layers {size: sub-hypergraph} (a multilayer view)
`simplicial_closure`	function	Downward closure: add every non-empty subset of every hyperedge (an abstract simplicial complex)
`from_cliques`	function	Hypergraph of the maximal cliques of a graph (a lossy inverse of the clique expansion)
`from_graph_edges`	function	A graph as a 2-uniform hypergraph (one hyperedge per graph edge)
`to_bipartite_frame`	function	Incidence list (node, edge, value) — the long form of the incidence matrix

A.3 `hatvis.io`: Interchange formats and converters

Name	Kind	Description
`to_hif`	function	Convert to a HIF dictionary (network-type, nodes, edges, incidences)
`from_hif`	function	Build a hypergraph from a HIF dictionary
`write_hif`	function	Write a HIF JSON file
`read_hif`	function	Read a HIF JSON file

Name	Kind	Description
write_hypergodot	function	Write a hypergraph in the HyperGodot text format (vertex labels must not contain spaces)
read_hypergodot	function	Read a HyperGodot input file (see parse_hypergodot)
parse_hypergodot	function	Parse the HyperGodot text format
read_spmf_itemsets	function	Read SPMF itemset output (1 4 7 #SUP: 25); the support becomes the hyperedge weight
read_spmf_rules	function	Read SPMF association rules (1 2 ==> 5 #SUP: 3 #CONF: 0.75) as a directed hypergraph
write_spmf_itemsets	function	Write hyperedges as SPMF itemsets (the weight is written as #SUP)
read_incidence_csv	function	Read a CSV file
write_incidence_csv	function	Write an incidence list (default) or the incidence matrix as CSV
read_incidence_excel	function	Read an incidence matrix from an Excel sheet (vertices in rows, hyperedges in columns)
to_hypernetx	function	Convert to a hypernetx.Hypergraph (weights as the edge property weight)
from_hypernetx	function	Convert from a hypernetx.Hypergraph (edge weights are read when available)
to_xgi	function	Convert to an xgi.Hypergraph (weights as the edge attribute weight)
from_xgi	function	Convert from an xgi.Hypergraph
to_hypergraphx	function	Convert to a hypergraphx.Hypergraph (hyperedge labels are not kept by Hypergraphx)
from_hypergraphx	function	Convert from a hypergraphx.Hypergraph
from_bipartite	function	Build a hypergraph from a bipartite graph (inverse of star_expansion)
to_networkx_bipartite	function	Alias of star_expansion

A.4 `hatvis.similarity`: Vertex and hyperedge similarity

Name	Kind	Description
vertex_similarity	function	Vertex similarity matrix $\mathbf{S}$ (rows and columns follow H.nodes)
edge_similarity	function	Hyperedge similarity matrix: the vertex similarity of the dual hypergraph
transitive_closure	function	Transitive closure of a similarity matrix (strongest-chain similarity)
dissimilarity	function	Dissimilarity matrix $\mathbf{\Delta} = \mathbf{1} - \mathbf{S}$ used by the similarity layouts

Name	Kind	Description
`hypergraph_distance`	function	Distance between two hypergraphs
`pairwise_profile_similarity`	function	Similarity between the rows of a non-negative profile matrix X

A.5 `hatvis.metrics`: Structural statistics and s-walks

Name	Kind	Description
`degree_distribution`	function	Number (or fraction) of vertices of each degree
`size_distribution`	function	Number (or fraction) of hyperedges of each size
`intersection_distribution`	function	Distribution of $\lvert e \cap f \rvert$ over unordered pairs of distinct hyperedges with a common vertex
`s_connected_components`	function	The s-connected components
`is_s_connected`	function	
`s_distance`	function	Length of the shortest s-walk between two hyperedges (or vertices); inf if none
`s_distance_matrix`	function	All-pairs s-distances (inf for unreachable pairs)
`s_diameter`	function	Largest finite s-distance (inf if the hypergraph is not s-connected)
`s_component_profile`	function	How connectivity changes with s: number of components, size of the largest one and diameter
`two_node_clustering`	function	Two-node clustering coefficient (Gallagher and Goldberg 2013)
`clique_clustering`	function	Local clustering coefficient of the clique expansion (counts every hyperedge as a clique)
`degree_assortativity`	function	Degree assortativity of a hypergraph (after Chodrow 2020)
`nestedness`	function	Inclusion statistics: how many hyperedges are proper subsets of another hyperedge
`simpliciality`	function	Simpliciality measures of Landry, Young and Eikmeier (2024)
`degree_entropy`	function	Shannon entropy (bits) of the degree distribution
`spectral_entropy`	function	von Neumann entropy of the normalized Laplacian, $-\sum_k \lambda_k/n \, \log_2(\lambda_k/n)$, a basis-free measure of structural heterogeneity

A.6 `hatvis.centrality`: Centrality measures

Name	Kind	Description
`degree_centrality`	function	Vertex degree, divided by the number of hyperedges (or the total weight) when normalized
`edge_strength`	function	Strength of a hyperedge: $w_e \sum_i b_{ie}$ (size for an unweighted hypergraph)
`s_betweenness`	function	The s-betweenness centrality (Aksoy et al. 2020)
`s_closeness`	function	The s-closeness centrality: reciprocal mean s-distance to the reachable elements
`s_harmonic_closeness`	function	Harmonic s-closeness: $HCC_s(e) = \frac{1}{\lvert E_s\rvert - 1} \sum_{f \in E_s, f \neq e} 1/d_s(e,f)$, with $E_s = \{e : \lvert e\rvert \geq s\}$ (unreachable pairs
`s_eigenvector`	function	The s-eigenvector centrality: principal eigenvector of the s-line graph
`average_s_centrality`	function	Average of an s-centrality over $s = 1, \ldots, s_{\max}$
`s_centrality_table`	function	Table of s-betweenness and harmonic s-closeness for several s values
`clique_eigenvector`	function	Clique-expansion eigenvector centrality (CEC; Benson 2019): Perron vector of $\mathbf{A}$
`z_eigenvector`	function	Z-eigenvector centrality (ZEC) of a uniform hypergraph (Benson 2019)
`h_eigenvector`	function	H-eigenvector centrality (HEC) of a uniform hypergraph (Benson 2019)
`node_edge_eigenvector`	function	Node and edge nonlinear eigenvector centralities (Tudisco and Higham 2021)
`pagerank`	function	PageRank of the hypergraph random walk
`hypercore_numbers`	function	Core numbers of the (k, m)-hyper-core decomposition (Mancastroppa et al. 2023)
`hypercoreness`	function	Hyper-coreness $R(i) = \sum_m C_m(i)/k_{\max}(m)$ (Mancastroppa et al. 2023)

A.7 `hatvis.spectral`: Random walks and spectral methods

Name	Kind	Description
`transition_matrix`	function	Row-stochastic transition matrix of a random walk on the vertices
`stationary_distribution`	function	Stationary distribution $\boldsymbol{\pi} = \boldsymbol{\pi}\mathbf{P}$ of a random walk
`spectral_embedding`	function	Spectral embedding of the vertices (Zhou et al. 2006)

Name	Kind	Description
`spectral_clustering`	function	Spectral clustering of the vertices with the hypergraph Laplacian (Zhou et al. 2006)
`fiedler_vector`	function	Eigenvector of the second smallest Laplacian eigenvalue (sign fixed: first entry non-negative)
`algebraic_connectivity`	function	Second smallest Laplacian eigenvalue (zero iff the hypergraph is disconnected)

A.8 `hatvis.community`: Modularity and community detection

Name	Kind	Description
`modularity`	function	Strict (all-or-nothing) hypergraph modularity
`graph_modularity`	function	Newman–Girvan modularity of a projection (clique expansion of the vertices)
`hypergraph_louvain`	function	Maximize the strict hypergraph modularity (modularity) by local moving and merging
`louvain`	function	Louvain communities of a projection (networkx implementation)
`label_propagation`	function	Label propagation through the incidence structure (after Raghavan et al. 2007)
`spectral_coclustering`	function	Simultaneous clustering of vertices and hyperedges (spectral co-clustering, Dhillon 2001)
`partition_sizes`	function	Number of members per community, largest first
`communities`	function	Partition as a list of member lists (community 0 first)

A.9 `hatvis.partition`: Hypergraph partitioning

Name	Kind	Description
`cut_metrics`	function	Quality of a vertex partition
`normalized_cut`	function	Normalized hypergraph cut of a vertex set S (Zhou et al. 2006)
`fm_bisection`	function	Balanced bisection minimizing the cut-net metric (Fiduccia and Mattheyses 1982)

Name	Kind	Description
recursive_bisection	function	k-way partition by recursive FM bisection (n_blocks a power of two is balanced best)

A.10 hatvis.patterns: Frequent patterns and association rules

Name	Kind	Description
transactions_to_hypergraph	function	One hyperedge per transaction (repeated transactions stay separate hyperedges)
hypergraph_to_transactions	function	Hyperedges as transactions
frequent_itemsets	function	Frequent itemsets of a transaction database or of a hypergraph
closed_itemsets	function	Closed itemsets: no proper superset has the same support (Pasquier et al. 1999)
maximal_itemsets	function	Maximal frequent itemsets: no proper superset is frequent
association_rules	function	Association rules $A \Rightarrow C$ from a table of frequent itemsets
itemset_hypergraph	function	Hypergraph whose hyperedges are itemsets (weights: support by default)
rule_hypergraph	function	Directed hypergraph of association rules: tail = antecedent, head = consequent
arhp_weights	function	Hyperedge weights of association-rule hypergraph partitioning (Han et al. 1997)
validated_hyperedges	function	Statistically validated hyperedges (after Musciotto, Battiston and Mantegna 2021)
inclusion_lattice	function	Hasse diagram of the inclusion order of a family of sets
formal_concepts	function	Formal concepts of the incidence relation (Ganter and Wille 1999)

A.11 hatvis.motifs: Higher-order motifs

Name	Kind	Description
h_motif_catalog	function	The 26 h-motifs: canonical region patterns and a short description
h_motif_pattern	function	h-motif identifier of three hyperedges (sets of vertices)

Name	Kind	Description
`h_motifs`	function	Number of instances of every h-motif among the connected triples of distinct hyperedges
`node_motifs`	function	Counts of the six connected node motifs of order three
`motif_zscores`	function	Compare motif counts with a null model

A.12 `hatvis.models`: Random hypergraph models

Name	Kind	Description
`uniform_random`	function	Uniform random hypergraph: m hyperedges, each a uniformly random k-subset of n vertices
`random_hypergraph`	function	Random hypergraph with a prescribed size sequence (each hyperedge a uniform random subset)
`chung_lu`	function	Chung–Lu hypergraph: vertices enter a hyperedge with probability proportional to their degree
`configuration_model`	function	Degree- and size-preserving randomization by pairwise reshuffles (Chodrow 2020)
`planted_partition`	function	Hypergraph stochastic block model with planted communities
`temporal_random`	function	Temporal hypergraph with tunable persistence: a fraction of the hyperedges is repeated from the last step

A.13 `hatvis.problems`: Combinatorial problems

Name	Kind	Description
`greedy_set_cover`	function	Greedy set cover (Chvátal 1979)
`set_cover`	function	Minimum-cost set cover as an integer program (solved with SciPy's HiGHS interface)
`greedy_hitting_set`	function	Greedy hitting set: the greedy set cover of the dual hypergraph
`hitting_set`	function	Minimum (cost) hitting set, i.e. minimum vertex cover / transversal, as an integer program

Name	Kind	Description
minimal_transversals	function	All minimal transversals (Berge's sequential algorithm)
transversal_hypergraph	function	The transversal (blocker) hypergraph $Tr(H)$ of the minimal transversals
maximum_matching	function	Maximum (weight) matching: pairwise disjoint hyperedges, as an integer program
greedy_coloring	function	Greedy vertex coloring
is_two_colorable	function	Decide Property B (2-colorability without monochromatic hyperedge) by backtracking
gyo_reduction	function	GYO reduction (Graham 1979; Yu and Özsoyoğlu 1979)
is_alpha_acyclic	function	True if the hypergraph is α-acyclic (see gyo_reduction)
join_tree	function	Join tree of an α-acyclic hypergraph
is_conformal	function	Conformality: every maximal clique of the clique expansion lies inside a hyperedge
has_helly_property	function	Helly property: every pairwise intersecting subfamily has a common vertex
b_reachable	function	Vertices B-reachable from sources (B-visit, Gallo et al. 1993)
shortest_hyperpaths	function	Minimum B-hyperpath distances from sources (SBT procedure of Gallo et al. 1993)

A.14 `hatvis.qfd`: Quality function deployment

Name	Kind	Description
qfd_hypergraph	function	Hypergraph of a house of quality: vertices = requirements, hyperedges = parameters
generalized_qfd_hypergraph	function	Generalized hypergraph of a house of quality (Abonyi and Czvetkó 2022)
correlation_hypergraph	function	Hypergraph of the roof: every parameter is a hyperedge of itself and the parameters it correlates with
qfd_select	function	Choose the development parameters by mixed-integer linear programming (Abonyi and Czvetkó 2022)
qfd_min_cost_assignment	function	QFD as a minimum-cost flow (assignment) problem
technical_importance	function	Classical QFD technical importance $t_k = \sum_i \lambda_i a_{ik}$ of every parameter

A.15 `hatvis.dynamics`: Dynamical processes

Name	Kind	Description
`diffusion`	function	Linear diffusion $\dot{\mathbf{x}} = -\mathbf{L}\mathbf{x}$ with a hypergraph Laplacian
`group_consensus`	function	Nonlinear group consensus (a generalization of the 3-body model of Neuhäuser et al. 2020)
`threshold_contagion`	function	Discrete-time SIS contagion through hyperedges with a threshold
`random_walk_trajectory`	function	Simulate a random walk on the vertices (see transition_matrix)

A.16 `hatvis.learning`: Semi-supervised learning and prediction

Name	Kind	Description
`propagation_operator`	function	Normalized propagation operator of the hypergraph
`label_spreading`	function	Semi-supervised classification on a hypergraph (Zhou et al. 2006)
`hgnn_features`	function	Propagate vertex features, $\mathbf{X} \leftarrow \mathbf{\Theta}^K\mathbf{X}$ (HGNN operator, linear)
`hgnn_classifier`	function	Simplified HGNN: hgnn_features + multinomial logistic regression trained on the labeled vertices
`hyperedge_score`	function	Plausibility score of a candidate vertex set as a hyperedge
`hyperedge_prediction`	function	Evaluate hyperedge prediction by hiding a fraction of the hyperedges

A.17 `hatvis.temporal`: Temporal and comparative analysis

Name	Kind	Description
`group_statistics`	function	Statistics of the hyperedges per group tag (edge_data[group])

Name	Kind	Description
turnover	function	Change between consecutive snapshots: new, persisting and dropped vertex sets and their Jaccard similarity
persistence	function	For every distinct vertex set: number of time stamps in which it occurs, first and last occurrence
difference	function	Union of two hypergraphs with edge_data["status"] = both / only first / only second
temporal_centrality	function	Apply a vertex measure (e.g. hatvis.degree_centrality) to every snapshot: time x vertices

A.18 hatvis.seriation: Seriation and biclustering

Name	Kind	Description
barycentric_order	function	Iterative barycentric order of the rows (axis=0) or columns (axis=1) of a matrix
correspondence_order	function	Row and column orders from the first axis of correspondence analysis
hierarchical_order	function	Leaf order of a hierarchical clustering of the rows (or columns), optionally with optimal leaf ordering
seriate	function	Row (vertex) and column (hyperedge) orders of the incidence matrix
seriation_quality	function	Compactness of an ordered incidence matrix
bicluster_blocks	function	Biclusters of the incidence matrix by spectral co-clustering

A.19 hatvis.layout: Layout algorithms

Name	Kind	Description
classical_mds	function	Classical (Torgerson) multidimensional scaling
sammon	function	Sammon mapping (Sammon 1969) of a dissimilarity matrix
layout_similarity	function	Similarity-preserving layout (HAT-VIS)
layout_force	function	Force-directed layout of HyperGodot (Ficsor et al. 2025)
layout_membership	function	Membership-aware layout (new in hatvis)

Name	Kind	Description
`layout_circular`	function	Vertices evenly on a circle; order="seriation" uses the barycentric order to shorten hyperedge spans
`layout_grid`	function	Vertices on a square grid (row-major, in vertex order or the given order)
`layout_radial`	function	Radial layout: the most central vertex (largest degree) in the middle, the others on rings by graph distance
`layout_spectral`	function	Coordinates from the two leading non-trivial Laplacian eigenvectors (Zhou et al. 2006)
`layout_bipartite`	function	Spring layout of the star expansion; hyperedges act as hidden attractor nodes
`edge_centroids`	function	Centroid of the member positions of every non-empty hyperedge
`layout_quality`	function	Quality of a layout for hypergraph drawing
`normalize_layout`	function	Scale coordinates to the unit box (keeping the aspect ratio) and return a label dict

A.20 `hatvis.interactive`: Interactive viewer

Name	Kind	Description
`to_html`	function	Write a self-contained interactive HTML view of a hypergraph and return the HTML text

A.21 `hatvis.datasets`: Data sets

Name	Kind	Description
`Bunch`	class	Dictionary with attribute access (like sklearn.utils.Bunch)
`hatvis_example`	function	The example hypergraph of the HAT-VIS paper (Czvetkó and Abonyi 2024, Table 2)
`icms_example`	function	Operators and activities (Nagy et al. 2022, Fig
`load_electric_car_qfd`	function	House of quality of an autonomous electric car, the case study of HAT-VIS
`load_manufacturing_cps`	function	Cyber-physical production example of HyperGodot (Ficsor et al. 2025, Tables 5–6)

Name	Kind	Description
load_wire_harness	function	Intelligent collaborative manufacturing space of a wire harness assembly benchmark
make_news_transactions	function	Synthetic news articles described by their themes (a stand-in for GDELT-type data)

A.22 hatvis.plotting: Plotting

Name	Kind	Description
PALETTE	constant	Okabe–Ito categorical palette; hyperedge or group i always gets PALETTE[i]
BRAND	constant	The Pannon blue (#285780) used for single-color figures
GRAY	constant	Text and neutral colors of the figures
INK	constant	Text and neutral colors of the figures
LINESTYLES	constant	Line styles that tell groups of hyperedges apart (HyperGodot convention)
MARKERS	constant	Marker cycle for vertex types
color_cycle	function	n distinct colors: the Okabe–Ito palette first, then tab20 hues
draw_hypergraph	function	Draw a hypergraph as vertices with hyperedge outlines
draw_hatvis	function	The HAT-VIS drawing (Czvetkó and Abonyi 2024)
draw_bipartite	function	Star-expansion (bipartite, extra-node) drawing: vertices as circles, hyperedges as squares
draw_clique	function	Clique expansion drawn as a weighted graph (line width ~ pair weight)
draw_line_graph	function	The s-line graph: hyperedges as nodes (size ~ $\lvert e\rvert$), links width ~ $\lvert e \cap f\rvert$
draw_circular	function	Circular drawing: vertices on a circle, every hyperedge as a hub with curved spokes
draw_primal_dual	function	Linked primal and dual views (after Qu, Zhang and Zhang 2022)
resolve_layout	function	Return pos if given, otherwise compute the named layout
plot_incidence	function	Incidence matrix as a heat map, optionally seriated and with highlighted biclusters
plot_paoh	function	Parallel Aggregated Ordered Hypergraph (PAOH) view (Valdivia et al. 2021)
plot_upset	function	UpSet plot (Lex et al. 2014) of a hypergraph
plot_house_of_quality	function	House of quality diagram: relationship matrix, importance column, technical importance row and roof

Name	Kind	Description
`plot_similarity_matrix`	function	Heat map of the vertex similarity matrix in a seriated order
`plot_distributions`	function	Degree and hyperedge-size distributions side by side
`plot_s_profile`	function	The *s-profile* (new in hatvis): how every hyperedge behaves as the overlap threshold *s* grows
`plot_ranking`	function	Horizontal bar chart of the largest values of a centrality (or any) Series
`plot_lattice`	function	Draw an inclusion lattice (Hasse diagram, inclusion_lattice) in levels by size
`plot_motif_profile`	function	Bar chart of motif significance (output of motif_zscores)
`plot_activity`	function	Vertex activity over time: the number of hyperedges of each vertex per time stamp (heat map)
`plot_s_walks`	function	Side-by-side *s*-line graphs of the vertices for several *s* (cf
`convex_hull`	function	Convex hull vertices in counter-clockwise order (Andrew's monotone chain)
`padded_hull`	function	Convex hull of circles of radius pad around the points (HyperGodot style)
`smooth_closed_curve`	function	Chaikin corner cutting of a closed polygon (rounded hull outlines)
`polygon_area`	function	Area of a simple polygon (shoelace formula)
`min_volume_ellipse`	function	Minimum-volume enclosing ellipse of a point set
`ellipse_points`	function	Boundary points (a closed curve) of the ellipse returned by min_volume_ellipse
`minimum_spanning_segments`	function	Edges of the minimum spanning tree of a point set (Prim's algorithm)
`isocontour`	function	Implicit outline of a point set (in the spirit of Bubble Sets, Collins et al. 2009)

APPENDIX

B



Migrating from HAT-VIS, HyperGodot, and other packages

> This appendix is for readers who know the MATLAB scripts of HAT-VIS, the HyperGodot viewer, or one of the Python packages HyperNetX, XGI, and Hypergraphx. It maps their functions and settings to `hatvis` and lists the chapter where each method is explained.

B.1 HAT-VIS

The HAT-VIS distribution [1] consists of MATLAB scripts that build a `Hypergraph` object of HAT [18] from an incidence matrix, reorder the matrix, compute the similarity of the vertices, embed it with `cmdscale`, and draw the hyperedges. Table B.1 gives the `hatvis` counterpart of every step. Two details matter when results are compared. The electric-car script embeds the dissimilarity of the *transitive* similarity (`D=1-simOT`), which is the default of `layout_similarity`; the mini example embeds the plain similarity (`D=1-simO`), which corresponds to `transitive=False`. The hyperedge trees of both scripts are minimum spanning trees on the dissimilarities $1 - s_{ij}$ of the plain similarity, as in `draw_hatvis`.

B.2 HyperGodot

HyperGodot [2] is an interactive application; its settings correspond to arguments of the `hatvis` layout and drawing functions (table B.2). The input files are read and

Table B.1. HAT-VIS script steps and their `hatvis` counterparts.

HAT-VIS (MATLAB)	`hatvis`	Chapter
`Hypergraph('IM', IM)`	`Hypergraph.from_incidence(IM)`	2
`HAT.uniformErdosRenyi(v, e, k)`	`models.uniform_random(n, m, k)`	9
`adjTensor`, `degreeTensor`, `laplacianTensor`	`adjacency_tensor`, `degree_tensor`, `laplacian_tensor`	2
`cliqueGraph`, `starGraph`	`clique_expansion`, `star_expansion`	2
barycentric sorting loops, `spy(IM(x,y))`	`seriate(H, "barycentric")`, `plot_incidence(H)`	3
`simO` (weighted Jaccard loop)	`vertex_similarity(H)`	5
`simOT` (max-product loop)	`transitive_closure(S, "max-product")`, `vertex_similarity(H, transitive=True)`	5
`cmdscale(D, 2)`	`layout.classical_mds(D)`, `layout_similarity(H)`	5
MST of the members per hyperedge	`draw_hatvis(H, mode="mst")`	5
bounding ellipses, convex hulls	`draw_hypergraph(H, style="ellipse"|"hull")`	4
`xlsread` of the house of quality; `A.*w`	`io.read_incidence_excel`, `qfd.qfd_hypergraph(R, importance)`, `datasets.load_electric_car_qfd()`	5

written unchanged, and `to_html` provides an interactive view in the browser with dragging, zooming, the vertex filter, group toggles, layout switching, adjustable padding and smoothness, and SVG export.

Table B.2. HyperGodot features and their `hatvis` counterparts.

HyperGodot	`hatvis`	Chapter
input file with `#WEIGHT` and `#GROUP` tags	`io.read_hypergodot`, `io.write_hypergodot`	2
force-directed layout (iterations, repulsion, spring length)	`layout_force(H, iterations=500, repulsion=250, spring_length=30)`	4
circular and radial layouts	`layout_circular`, `layout_radial`	4
convex-hull hyperedges with padding	`draw_hypergraph(H, style="hull", pad, pad_step)`	4
groups shown by line styles	`draw_hypergraph(H, group="group")`	4
line widths by weight	`draw_hypergraph(H, width_by="weight")`	4
vertex filter	`draw_hypergraph(H, highlight=v)`	4
interactive window, image export	`to_html(H, path)`; Matplotlib `savefig`	4
comparison of system variants	`temporal.difference`, `temporal.group_statistics`	12

B.3 HyperNetX, XGI, and Hypergraphx

The three Python packages are reached through converters and through the Hypergraph Interchange Format (section 2.3). Table B.3 lists the conversions and the `hatvis` functions that correspond to frequently used concepts of these packages; the names of the functions in the other packages change between versions, so the table refers to the concepts and the chapters rather than to their exact spelling.

Table B.3. Interoperation with HyperNetX, XGI, and Hypergraphx.

Concept or object	`hatvis`	Chapter
HyperNetX hypergraph	`io.to_hypernetx(H)`, `io.from_hypernetx(h)`	2
XGI hypergraph	`io.to_xgi(H)`, `io.from_xgi(h)`	2
Hypergraphx hypergraph	`io.to_hypergraphx(H)`, `io.from_hypergraphx(h)`	2
HIF file (all three)	`io.write_hif`, `io.read_hif`	2
s-line graphs, s-components, s-distance	`s_line_graph`, `s_connected_components`, `s_distance`	6
s-betweenness, s-closeness	`s_betweenness`, `s_closeness`, `s_harmonic_closeness`	6
clustering coefficients	`metrics.two_node_clustering`, `metrics.clique_clustering`	9
CEC, ZEC, HEC, node–edge centralities	`clique_eigenvector`, `z_eigenvector`, `h_eigenvector`, `node_edge_eigenvector`	6
motifs	`motifs.h_motifs`, `motifs.node_motifs`, `motifs.motif_zscores`	9
statistically validated hyperedges	`patterns.validated_hyperedges`	8
configuration model	`models.configuration_model`	9
simpliciality	`simpliciality`	9
drawing	`draw_hypergraph`, `draw_hatvis`, `to_html`	4

APPENDIX

C



The example gallery

Every method of the toolbox has at least one runnable example script. The scripts load or generate their data, explain the method in their comments, and print the numbers quoted in the text. 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 Representations and interchange

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

Script	Topic	Figure
`plot_01_quickstart.py`	A first hypergraph: incidence matrix, dual and degrees	1.3
`plot_02_projection_loss.py`	What the clique expansion forgets	2.2
`plot_03_expansions_and_spectra.py`	Expansions, line graphs and Laplacian spectra	2.3
`plot_04_interchange.py`	Interchange: HIF, HyperGodot files and other Python libraries	2.5

C.2 Matrix views

Folder `examples/02_matrix_views/`. The methods are described in chapter 3.

Script	Topic	Figure
`plot_05_seriation.py`	Seriation of the incidence matrix	3.2
`plot_06_biclusters.py`	Biclusters of the manufacturing space	3.3
`plot_07_paoh.py`	PAOH: a dynamic hypergraph without clutter	3.4
`plot_08_upset.py`	UpSet view of frequent theme combinations	3.5

C.3 Node-link and set views

Folder `examples/03_node_link/`. The methods are described in chapter 4.

Script	Topic	Figure
`plot_09_layouts.py`	Six layouts of the same hypergraph	4.2
`plot_10_hyperedge_styles.py`	Seven ways to draw a hyperedge	4.3
`plot_11_groups.py`	Comparing groups: human and machine work	4.4
`plot_12_primal_dual.py`	Linked primal and dual views	4.5
`plot_13_circular_and_interactive.py`	Circular drawing and the interactive HTML viewer	4.6

C.4 Similarity-based visualization

Folder `examples/04_similarity/`. The methods are described in chapter 5.

Script	Topic	Figure
`plot_14_hatvis_pipeline.py`	The HAT-VIS pipeline step by step	5.2
`plot_15_electric_car.py`	HAT-VIS case study: developing an autonomous electric car	5.3
`plot_16_membership_layout.py`	Membership-aware layout: similarity with fewer intrusions	5.4

C.5 Walks, connectivity, and centrality

Folder `examples/05_structure/`. The methods are described in chapter 6.

Script	Topic	Figure
`plot_17_s_walks.py`	s-walks: how strongly are hyperedges connected?	6.2
`plot_18_wire_harness_centrality.py`	Critical elements of the manufacturing space	6.4
`plot_19_centrality_family.py`	A family of hypergraph centralities	6.5
`plot_20_hypercores.py`	Hyper-cores	6.6

C.6 Clustering and partitioning

Folder `examples/06_clustering/`. The methods are described in chapter 7.

Script	Topic	Figure
`plot_21_planted_communities.py`	Community detection on a planted-partition hypergraph	7.2
`plot_22_cell_formation.py`	Forming manufacturing cells from the hypergraph	7.3
`plot_23_partitioning.py`	Balanced partitioning and normalized cuts	7.4
`plot_24_coclustering.py`	Co-clustering vertices and hyperedges	7.5

C.7 Frequent patterns and association rules

Folder `examples/07_patterns/`. The methods are described in chapter 8.

Script	Topic	Figure
`plot_25_itemset_hypergraph.py`	Frequent itemsets as a hypergraph	8.2
`plot_26_association_rules.py`	Association rules as a directed hypergraph	8.4
`plot_27_validated_hyperedges.py`	Statistically validated hyperedges	8.5

C.8 Higher-order structure and null models

Folder `examples/08_higher_order/`. The methods are described in chapter 9.

Script	Topic	Figure
`plot_28_h_motifs.py`	h-motifs and their significance	9.2
`plot_29_simpliciality.py`	Simpliciality and nestedness: hypergraph or simplicial complex?	9.3
`plot_30_null_models.py`	Null models: what is structure and what is chance?	9.4

C.9 Combinatorial problem solving

Folder `examples/09_problems/`. The methods are described in chapter 10.

Script	Topic	Figure
`plot_31_sensor_placement.py`	Covering and hitting: minimal monitoring sets	10.2
`plot_32_acyclicity.py`	Acyclic hypergraphs and join trees	10.3
`plot_33_directed_assembly.py`	Directed hypergraphs: assembly feasibility and cost	10.4
`plot_34_qfd_optimization.py`	QFD: which specifications should be developed?	10.5

C.10 Dynamics and learning

Folder `examples/10_dynamics_learning/`. The methods are described in chapter 11.

Script	Topic	Figure
`plot_35_diffusion_consensus.py`	Diffusion and group consensus	11.2
`plot_36_contagion.py`	Threshold contagion through groups	11.3
`plot_37_learning.py`	Semi-supervised learning and hyperedge prediction	11.4

C.11 Temporal and comparative analysis

Folder `examples/11_temporal/`. The methods are described in chapter 12.

Script	Topic	Figure
`plot_38_evolving_narratives.py`	Evolving narratives in a temporal hypergraph	12.2
`plot_39_comparing_systems.py`	Comparing two versions of a production system	12.3

Bibliography

[1] Tímea Czvetkó and János Abonyi. "Version [1.0]- HAT-VIS — A MATLAB-based hypergraph visualization tool". In: *SoftwareX* 28 (2024), p. 101963. DOI: 10.1016/j.softx.2024.101963.

[2] Attila Ficsor, Tímea Czvetkó, Viktor Sebestyén, and János Abonyi. "HyperGodot: Interactive hypergraph visualization tool". In: *SoftwareX* 31 (2025), p. 102315. DOI: 10.1016/j.softx.2025.102315.

[3] János Abonyi and Tímea Czvetkó. "Hypergraph and network flow-based quality function deployment". In: *Heliyon* 8.12 (2022), e12263. DOI: 10.1016/j.heliyon.2022.e12263.

[4] László Nagy, Tamás Ruppert, Andreas Löcklin, and János Abonyi. "Hypergraph-based analysis and design of intelligent collaborative manufacturing space". In: *Journal of Manufacturing Systems* 65 (2022), pp. 88–103. DOI: 10.1016/j.jmsy.2022.08.001.

[5] 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.

[6] 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.

[7] Wes McKinney. "Data structures for statistical computing in Python". In: *Proceedings of the 9th Python in Science Conference*. 2010, pp. 56–61.

[8] 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 (SciPy 2008)*. 2008, pp. 11–15.

[9] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, et al. "Scikit-learn: Machine learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[10] 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.

[11] Federico Battiston et al. "Networks beyond pairwise interactions: Structure and dynamics". In: *Physics Reports* 874 (2020), pp. 1–92. DOI: 10.1016/j.physrep.2020.05.004.

[12] Federico Battiston et al. “The physics of higher-order interactions in complex systems”. In: *Nature Physics* 17 (2021), pp. 1093–1098. DOI: 10.1038/s41567-021-01371-4.

[13] Christian Bick, Elizabeth Gross, Heather A. Harrington, and Michael T. Schaub. “What are higher-order networks?” In: *SIAM Review* 65.3 (2023), pp. 686–731. DOI: 10.1137/21M1414024.

[14] Leo Torres, Ann S. Blevins, Danielle Bassett, and Tina Eliassi-Rad. “The why, how, and when of representations for complex systems”. In: *SIAM Review* 63.3 (2021), pp. 435–485. DOI: 10.1137/20M1355896.

[15] Claude Berge. *Graphs and Hypergraphs*. Amsterdam: North-Holland, 1973.

[16] Claude Berge. *Hypergraphs: Combinatorics of Finite Sets*. Vol. 45. North-Holland Mathematical Library. Amsterdam: North-Holland, 1989.

[17] Alain Bretto. *Hypergraph Theory: An Introduction*. Mathematical Engineering. Cham: Springer, 2013. DOI: 10.1007/978-3-319-00080-0.

[18] Joshua Pickard et al. “HAT: Hypergraph analysis toolbox”. In: *PLOS Computational Biology* 19.6 (2023), e1011190. DOI: 10.1371/journal.pcbi.1011190.

[19] Amit Surana, Can Chen, and Indika Rajapakse. “Hypergraph similarity measures”. In: *IEEE Transactions on Network Science and Engineering* 10.2 (2023), pp. 658–674. DOI: 10.1109/TNSE.2022.3217185.

[20] Brenda Praggastis et al. “HyperNetX: A Python package for modeling complex network data as hypergraphs”. In: *Journal of Open Source Software* 9.95 (2024), p. 6016. DOI: 10.21105/joss.06016.

[21] Sinan G. Aksoy, Cliff Joslyn, Carlos Ortiz Marrero, Brenda Praggastis, and Emilie Purvine. “Hypernetwork science via high-order hypergraph walks”. In: *EPJ Data Science* 9 (2020), p. 16. DOI: 10.1140/epjds/s13688-020-00231-0.

[22] Nicholas W. Landry et al. “XGI: A Python package for higher-order interaction networks”. In: *Journal of Open Source Software* 8.85 (2023), p. 5162. DOI: 10.21105/joss.05162.

[23] Quintino Francesco Lotito et al. “Hypergraphx: a library for higher-order network analysis”. In: *Journal of Complex Networks* 11.3 (2023), cnad019. DOI: 10.1093/comnet/cnad019.

[24] Bodian Ye et al. “EasyHypergraph: an open-source software for fast and memory-saving analysis and learning of higher-order networks”. In: *Humanities and Social Sciences Communications* 12 (2025), p. 1291. DOI: 10.1057/s41599-025-05180-5.

[25] Yue Gao, Yifan Feng, Shuyi Ji, and Rongrong Ji. “HGNN$^+$: General hypergraph neural networks”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45.3 (2023), pp. 3181–3199. DOI: 10.1109/TPAMI.2022.3182052.

[26] Martín Coll et al. “HIF: The hypergraph interchange format for higher-order networks”. In: *Network Science* 13 (2025), e21. DOI: 10.1017/nws.2025.10018.

[27] Masataka Okabe and Kei Ito. *Color Universal Design (CUD): How to make figures and presentations that are friendly to colorblind people*. 2008. URL: https://jfly.uni-koeln.de/color/.

[28] Qi Huangfu and J. A. Julian Hall. "Parallelizing the dual revised simplex method". In: *Mathematical Programming Computation* 10 (2018), pp. 119–142. DOI: 10.1007/s12532-017-0130-5.

[29] Geon Lee, Fanchen Bu, Tina Eliassi-Rad, and Kijung Shin. "A survey on hypergraph mining: Patterns, tools, and generators". In: *ACM Computing Surveys* (2025). DOI: 10.1145/3719002.

[30] Bilal Alsallakh, Luana Micallef, Wolfgang Aigner, Helwig Hauser, Silvia Miksch, and Peter Rodgers. "The state-of-the-art of set visualization". In: *Computer Graphics Forum* 35.1 (2016), pp. 234–260. DOI: 10.1111/cgf.12722.

[31] Maximilian T. Fischer, Alexander Frings, Daniel A. Keim, and Daniel Seebacher. "Towards a survey on static and dynamic hypergraph visualizations". In: *2021 IEEE Visualization Conference (VIS), Short Papers*. 2021, pp. 81–85.

[32] Hyung Lee-Kwang and Keon-Myung Lee. "Fuzzy hypergraph and fuzzy partition". In: *IEEE Transactions on Systems, Man, and Cybernetics* 25.1 (1995), pp. 196–201. DOI: 10.1109/21.362951.

[33] Giorgio Gallo, Giustino Longo, Stefano Pallottino, and Sang Nguyen. "Directed hypergraphs and applications". In: *Discrete Applied Mathematics* 42.2–3 (1993), pp. 177–201. DOI: 10.1016/0166-218X(93)90045-P.

[34] Giorgio Ausiello and Luigi Laura. "Directed hypergraphs: Introduction and fundamental algorithms—A survey". In: *Theoretical Computer Science* 658 (2017), pp. 293–306. DOI: 10.1016/j.tcs.2016.03.016.

[35] Dengyong Zhou, Jiayuan Huang, and Bernhard Schölkopf. "Learning with hypergraphs: Clustering, classification, and embedding". In: *Advances in Neural Information Processing Systems 19*. MIT Press, 2007, pp. 1601–1608.

[36] Sameer Agarwal, Kristin Branson, and Serge Belongie. "Higher order learning with graphs". In: *Proceedings of the 23rd International Conference on Machine Learning (ICML)*. 2006, pp. 17–24. DOI: 10.1145/1143844.1143847.

[37] Marianna Bolla. "Spectra, Euclidean representations and clusterings of hypergraphs". In: *Discrete Mathematics* 117.1–3 (1993), pp. 19–39. DOI: 10.1016/0012-365X(93)90322-K.

[38] J. A. Rodríguez. "On the Laplacian eigenvalues and metric parameters of hypergraphs". In: *Linear and Multilinear Algebra* 50.1 (2002), pp. 1–14. DOI: 10.1080/03081080290011692.

[39] Koby Hayashi, Sinan G. Aksoy, Cheong Hee Park, and Haesun Park. "Hypergraph random walks, Laplacians, and clustering". In: *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM)*. 2020, pp. 495–504. DOI: 10.1145/3340531.3412034.

[40] Uthsav Chitra and Benjamin J. Raphael. "Random walks on hypergraphs with edge-dependent vertex weights". In: *Proceedings of the 36th International Conference on Machine Learning*. Vol. 97. Proceedings of Machine Learning Research. 2019, pp. 1172–1181.

[41] Maxime Lucas, Giulia Cencetti, and Federico Battiston. "Multiorder Laplacian for synchronization in higher-order networks". In: *Physical Review Research* 2 (2020), p. 033410. DOI: 10.1103/PhysRevResearch.2.033410.

[42] Joshua Cooper and Aaron Dutle. "Spectra of uniform hypergraphs". In: *Linear Algebra and its Applications* 436.9 (2012), pp. 3268–3292. DOI: 10.1016/j.laa.2011.11.018.

[43] 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, Part III*. Vol. 9853. Lecture Notes in Computer Science. Springer, 2016, pp. 36–40. DOI: 10.1007/978-3-319-46131-1_8.

[44] Innar Liiv. "Seriation and matrix reordering methods: An historical overview". In: *Statistical Analysis and Data Mining* 3.2 (2010), pp. 70–91. DOI: 10.1002/sam.10071.

[45] 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.

[46] Stefan Niermann. "Optimizing the ordering of tables with evolutionary computation". In: *The American Statistician* 59.1 (2005), pp. 41–46. DOI: 10.1198/000313005X22770.

[47] William T. McCormick, Paul J. Schweitzer, and Thomas W. White. "Problem decomposition and data reorganization by a clustering technique". In: *Operations Research* 20.5 (1972), pp. 993–1009. DOI: 10.1287/opre.20.5.993.

[48] Michael Hahsler, Kurt Hornik, and Christian Buchta. "Getting things in order: An introduction to the R package seriation". In: *Journal of Statistical Software* 25.3 (2008), pp. 1–34. DOI: 10.18637/jss.v025.i03.

[49] M. O. Hill. "Correspondence analysis: A neglected multivariate method". In: *Journal of the Royal Statistical Society, Series C (Applied Statistics)* 23.3 (1974), pp. 340–354. DOI: 10.2307/2347127.

[50] Ziv Bar-Joseph, David K. Gifford, and Tommi S. Jaakkola. "Fast optimal leaf ordering for hierarchical clustering". In: *Bioinformatics* 17.suppl_1 (2001), S22–S29. DOI: 10.1093/bioinformatics/17.suppl_1.S22.

[51] Inderjit S. Dhillon. "Co-clustering documents and words using bipartite spectral graph partitioning". In: *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2001, pp. 269–274. DOI: 10.1145/502512.502550.

[52] 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.

[53] Paola Valdivia, Paolo Buono, Catherine Plaisant, Nicole Dufournaud, and Jean-Daniel Fekete. "Analyzing dynamic hypergraphs with parallel aggregated ordered hypergraph visualization". In: *IEEE Transactions on Visualization and Computer Graphics* 27.1 (2021), pp. 1–13. DOI: 10.1109/TVCG.2019.2933196.

[54] Kalev Leetaru and Philip A. Schrodt. "GDELT: Global data on events, location, and tone, 1979–2012". In: *ISA Annual Convention*. 2013.

[55] Erkki Mäkinen. "How to draw a hypergraph". In: *International Journal of Computer Mathematics* 34.3–4 (1990), pp. 177–185. DOI: 10.1080/00207169008803875.

[56] François Bertault and Peter Eades. "Drawing hypergraphs in the subset standard". In: *Graph Drawing (GD 2000)*. Vol. 1984. Lecture Notes in Computer Science. Springer, 2001, pp. 164–169. DOI: 10.1007/3-540-44541-2_15.

[57] David S. Johnson and Henry O. Pollak. "Hypergraph planarity and the complexity of drawing Venn diagrams". In: *Journal of Graph Theory* 11.3 (1987), pp. 309–325. DOI: 10.1002/jgt.3190110306.

[58] Thomas M. J. Fruchterman and Edward M. Reingold. "Graph drawing by force-directed placement". In: *Software: Practice and Experience* 21.11 (1991), pp. 1129–1164. DOI: 10.1002/spe.4380211102.

[59] Naheed Anjum Arafat and Stéphane Bressan. "Hypergraph drawing by force-directed placement". In: *Database and Expert Systems Applications (DEXA 2017)*. Vol. 10439. Lecture Notes in Computer Science. Springer, 2017, pp. 387–394. DOI: 10.1007/978-3-319-64471-4_31.

[60] 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.

[61] A. M. Andrew. "Another efficient algorithm for convex hulls in two dimensions". In: *Information Processing Letters* 9.5 (1979), pp. 216–219. DOI: 10.1016/0020-0190(79)90072-3.

[62] George Merrill Chaikin. "An algorithm for high-speed curve generation". In: *Computer Graphics and Image Processing* 3.4 (1974), pp. 346–349. DOI: 10.1016/0146-664X(74)90028-8.

[63] Christopher Collins, Gerald Penn, and Sheelagh Carpendale. "Bubble Sets: Revealing set relations with isocontours over existing visualizations". In: *IEEE Transactions on Visualization and Computer Graphics* 15.6 (2009), pp. 1009–1016. DOI: 10.1109/TVCG.2009.122.

[64] Leonid G. Khachiyan. "Rounding of polytopes in the real number model of computation". In: *Mathematics of Operations Research* 21.2 (1996), pp. 307–320. DOI: 10.1287/moor.21.2.307.

[65] Michael J. Todd and E. Alper Yıldırım. "On Khachiyan's algorithm for the computation of minimum-volume enclosing ellipsoids". In: *Discrete Applied Mathematics* 155.13 (2007), pp. 1731–1744. DOI: 10.1016/j.dam.2007.02.013.

[66] Botong Qu, Eugene Zhang, and Yue Zhang. "Automatic polygon layout for primal-dual visualization of hypergraphs". In: *IEEE Transactions on Visualization and Computer Graphics* 28.1 (2022), pp. 633–642. DOI: 10.1109/TVCG.2021.3114759.

[67] Joseph B. Kruskal. "On the shortest spanning subtree of a graph and the traveling salesman problem". In: *Proceedings of the American Mathematical Society* 7.1 (1956), pp. 48–50. DOI: 10.1090/S0002-9939-1956-0078686-7.

[68] Peter Rodgers. "A survey of Euler diagrams". In: *Journal of Visual Languages & Computing* 25.3 (2014), pp. 134–155. DOI: 10.1016/j.jvlc.2013.08.006.

[69] Peter Oliver, Eugene Zhang, and Yue Zhang. "Scalable hypergraph visualization". In: *IEEE Transactions on Visualization and Computer Graphics* 30.1 (2024), pp. 595–605. DOI: 10.1109/TVCG.2023.3326599.

[70] Peter Oliver, Eugene Zhang, and Yue Zhang. "Structure-aware simplification for hypergraph visualization". In: *IEEE Transactions on Visualization and Computer Graphics* 31.1 (2025), pp. 667–676. DOI: 10.1109/TVCG.2024.3456367.

[71] Youjia Zhou, Archit Rathore, Emilie Purvine, and Bei Wang. "Topological simplifications of hypergraphs". In: *IEEE Transactions on Visualization and Computer Graphics* 29.7 (2023), pp. 3209–3225. DOI: 10.1109/TVCG.2022.3153895.

[72] Paul Jaccard. "Étude comparative de la distribution florale dans une portion des Alpes et du Jura". In: *Bulletin de la Société Vaudoise des Sciences Naturelles* 37 (1901), pp. 547–579.

[73] Robert W. Floyd. "Algorithm 97: Shortest path". In: *Communications of the ACM* 5.6 (1962), p. 345. DOI: 10.1145/367766.368168.

[74] Lotfi A. Zadeh. "Similarity relations and fuzzy orderings". In: *Information Sciences* 3.2 (1971), pp. 177–200. DOI: 10.1016/S0020-0255(71)80005-1.

[75] Warren S. Torgerson. "Multidimensional scaling: I. Theory and method". In: *Psychometrika* 17.4 (1952), pp. 401–419. DOI: 10.1007/BF02288916.

[76] John W. Sammon. "A nonlinear mapping for data structure analysis". In: *IEEE Transactions on Computers* C-18.5 (1969), pp. 401–409. DOI: 10.1109/T-C.1969.222678.

[77] Jan de Leeuw and Patrick Mair. "Multidimensional scaling using majorization: SMACOF in R". In: *Journal of Statistical Software* 31.3 (2009), pp. 1–30. DOI: 10.18637/jss.v031.i03.

[78] Emden R. Gansner, Yehuda Koren, and Stephen North. "Graph drawing by stress majorization". In: *Graph Drawing (GD 2004)*. Vol. 3383. Lecture Notes in Computer Science. Springer, 2005, pp. 239–250. DOI: 10.1007/978-3-540-31843-9_25.

[79] Tim Dwyer, Yehuda Koren, and Kim Marriott. "IPSep-CoLa: An incremental procedure for separation constraint layout of graphs". In: *IEEE Transactions on Visualization and Computer Graphics* 12.5 (2006), pp. 821–828. DOI: 10.1109/TVCG.2006.156.

[80] Yoji Akao. *Quality Function Deployment: Integrating Customer Requirements into Product Design*. Cambridge, MA: Productivity Press, 1990.

[81] Lai-Kow Chan and Ming-Lu Wu. "Quality function deployment: A literature review". In: *European Journal of Operational Research* 143.3 (2002), pp. 463–497. DOI: 10.1016/S0377-2217(02)00178-9.

[82] John R. Hauser and Don Clausing. "The house of quality". In: *Harvard Business Review* 66.3 (1988), pp. 63–73.

[83] Francesco Tudisco and Desmond J. Higham. "Node and edge nonlinear eigenvector centrality for hypergraphs". In: *Communications Physics* 4 (2021), p. 201. DOI: 10.1038/s42005-021-00704-2.

[84] Helcio Felippe, Alec Kirkley, and Federico Battiston. *Information theory for hypergraph similarity*. arXiv preprint. 2025.

[85] Manh Tuan Do, Se-eun Yoon, Bryan Hooi, and Kijung Shin. "Structural patterns and generative models of real-world hypergraphs". In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2020, pp. 176–186. DOI: 10.1145/3394486.3403060.

[86] Linton C. Freeman. "A set of measures of centrality based on betweenness". In: *Sociometry* 40.1 (1977), pp. 35–41. DOI: 10.2307/3033543.

[87] Massimo Marchiori and Vito Latora. "Harmony in the small-world". In: *Physica A* 285.3–4 (2000), pp. 539–546. DOI: 10.1016/S0378-4371(00)00311-3.

[88] Timoteo Carletti, Federico Battiston, Giulia Cencetti, and Duccio Fanelli. "Random walks on hypergraphs". In: *Physical Review E* 101 (2020), p. 022308. DOI: 10.1103/PhysRevE.101.022308.

[89] Sergey Brin and Lawrence Page. "The anatomy of a large-scale hypertextual Web search engine". In: *Computer Networks and ISDN Systems* 30.1–7 (1998), pp. 107–117. DOI: 10.1016/S0169-7552(98)00110-X.

[90] Austin R. Benson. "Three hypergraph eigenvector centralities". In: *SIAM Journal on Mathematics of Data Science* 1.2 (2019), pp. 293–312. DOI: 10.1137/18M1203031.

[91] Michael Ng, Liqun Qi, and Guanglu Zhou. "Finding the largest eigenvalue of a nonnegative tensor". In: *SIAM Journal on Matrix Analysis and Applications* 31.3 (2009), pp. 1090–1099. DOI: 10.1137/09074838X.

[92] Tamara G. Kolda and Jackson R. Mayo. "Shifted power method for computing tensor eigenpairs". In: *SIAM Journal on Matrix Analysis and Applications* 32.4 (2011), pp. 1095–1124. DOI: 10.1137/100801482.

[93] Marco Mancastroppa, Iacopo Iacopini, Giovanni Petri, and Alain Barrat. "Hyper-cores promote localization and efficient seeding in higher-order processes". In: *Nature Communications* 14 (2023), p. 6223. DOI: 10.1038/s41467-023-41887-2.

[94] Ernesto Estrada and Juan A. Rodríguez-Velázquez. "Subgraph centrality and clustering in complex hyper-networks". In: *Physica A* 364 (2006), pp. 581–594. DOI: 10.1016/j.physa.2005.12.002.

[95] Nate Veldt, Austin R. Benson, and Jon Kleinberg. "Hypergraph cuts with general splitting functions". In: *SIAM Review* 64.3 (2022), pp. 650–685. DOI: 10.1137/20M1321048.

[96] M. E. J. Newman. "Modularity and community structure in networks". In: *Proceedings of the National Academy of Sciences* 103.23 (2006), pp. 8577–8582. DOI: 10.1073/pnas.0601602103.

[97] Bogumił Kamiński, Valérie Poulin, Paweł Prałat, Przemysław Szufel, and François Théberge. "Clustering via hypergraph modularity". In: *PLoS ONE* 14.11 (2019), e0224307. DOI: 10.1371/journal.pone.0224307.

[98] Philip S. Chodrow, Nate Veldt, and Austin R. Benson. "Generative hypergraph clustering: From blockmodels to modularity". In: *Science Advances* 7.28 (2021), eabh1303. DOI: 10.1126/sciadv.abh1303.

[99] 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), P10008. DOI: 10.1088/1742-5468/2008/10/P10008.

[100] Tarun Kumar, Sankaran Vaidyanathan, Harini Ananthapadmanabhan, Srinivasan Parthasarathy, and Balaraman Ravindran. "Hypergraph clustering by iteratively reweighted modularity maximization". In: *Applied Network Science* 5 (2020), p. 52. DOI: 10.1007/s41109-020-00300-3.

[101] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. "Near linear time algorithm to detect community structures in large-scale networks". In: *Physical Review E* 76 (2007), p. 036106. DOI: 10.1103/PhysRevE.76.036106.

[102] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. "On spectral clustering: Analysis and an algorithm". In: *Advances in Neural Information Processing Systems 14*. MIT Press, 2002, pp. 849–856.

[103] Martina Contisciani, Federico Battiston, and Caterina De Bacco. "Inference of hyperedges and overlapping communities in hypergraphs". In: *Nature Communications* 13 (2022), p. 7229. DOI: 10.1038/s41467-022-34714-7.

[104] Nicolò Ruggeri, Martina Contisciani, Federico Battiston, and Caterina De Bacco. "Community detection in large hypergraphs". In: *Science Advances* 9.28 (2023), eadg9159. DOI: 10.1126/sciadv.adg9159.

[105] George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. "Multilevel hypergraph partitioning: Applications in VLSI domain". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 7.1 (1999), pp. 69–79. DOI: 10.1109/92.748202.

[106] Sebastian Schlag, Tobias Heuer, Lars Gottesbüren, Yaroslav Akhremtsev, Christian Schulz, and Peter Sanders. "High-quality hypergraph partitioning". In: *ACM Journal of Experimental Algorithmics* 27 (2023), p. 1.9. DOI: 10.1145/3529090.

[107] Richard M. Karp. "Reducibility among combinatorial problems". In: *Complexity of Computer Computations*. Ed. by Raymond E. Miller and James W. Thatcher. New York: Plenum Press, 1972, pp. 85–103. DOI: 10.1007/978-1-4684-2001-2_9.

[108] C. M. Fiduccia and R. M. Mattheyses. "A linear-time heuristic for improving network partitions". In: *Proceedings of the 19th Design Automation Conference*. 1982, pp. 175–181. DOI: 10.1109/DAC.1982.1585498.

[109] Lawrence Hubert and Phipps Arabie. "Comparing partitions". In: *Journal of Classification* 2.1 (1985), pp. 193–218. DOI: 10.1007/BF01908075.

[110] Rakesh Agrawal and Ramakrishnan Srikant. "Fast algorithms for mining association rules". In: *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB)*. 1994, pp. 487–499.

[111] Mohammed J. Zaki. "Scalable algorithms for association mining". In: *IEEE Transactions on Knowledge and Data Engineering* 12.3 (2000), pp. 372–390. DOI: 10.1109/69.846291.

[112] 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.

[113] Bernhard Ganter and Rudolf Wille. *Formal Concept Analysis: Mathematical Foundations*. Berlin: Springer, 1999. DOI: 10.1007/978-3-642-59830-2.

[114] Pang-Ning Tan, Vipin Kumar, and Jaideep Srivastava. "Selecting the right objective measure for association analysis". In: *Information Systems* 29.4 (2004), pp. 293–313. DOI: 10.1016/S0306-4379(03)00072-3.

[115] Tianhang Zhang. "Association rules". In: *Knowledge Discovery and Data Mining. Current Issues and New Applications (PAKDD 2000)*. Vol. 1805. Lecture Notes in Computer Science. Springer, 2000, pp. 245–256. DOI: 10.1007/3-540-45571-X_31.

[116] Eui-Hong Han, George Karypis, Vipin Kumar, and Bamshad Mobasher. "Clustering based on association rule hypergraphs". In: *Proceedings of the SIGMOD'97 Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD)*. 1997, pp. 9–13.

[117] Eui-Hong Han, George Karypis, Vipin Kumar, and Bamshad Mobasher. "Hypergraph based clustering in high-dimensional data sets: A summary of results". In: *IEEE Data Engineering Bulletin* 21.1 (1998), pp. 15–22.

[118] Federico Musciotto, Federico Battiston, and Rosario N. Mantegna. "Detecting informative higher-order interactions in statistically validated hypergraphs". In: *Communications Physics* 4 (2021), p. 218. DOI: 10.1038/s42005-021-00710-4.

[119] Yoav Benjamini and Yosef Hochberg. "Controlling the false discovery rate: A practical and powerful approach to multiple testing". In: *Journal of the Royal Statistical Society, Series B* 57.1 (1995), pp. 289–300. DOI: 10.1111/j.2517-6161.1995.tb02031.x.

[120] Philip S. Chodrow. "Configuration models of random hypergraphs". In: *Journal of Complex Networks* 8.3 (2020), cnaa018. DOI: 10.1093/comnet/cnaa018.

[121] Fan Chung and Linyuan Lu. "Connected components in random graphs with given expected degree sequences". In: *Annals of Combinatorics* 6 (2002), pp. 125–145. DOI: 10.1007/PL00012580.

[122] Sinan G. Aksoy, Tamara G. Kolda, and Ali Pinar. "Measuring and modeling bipartite graphs with community structure". In: *Journal of Complex Networks* 5.4 (2017), pp. 581–603. DOI: 10.1093/comnet/cnx001.

[123] Suzanne R. Gallagher and Debra S. Goldberg. "Clustering coefficients in protein interaction hypernetworks". In: *Proceedings of the International Conference on Bioinformatics, Computational Biology and Biomedical Informatics (ACM BCB)*. 2013, pp. 552–560. DOI: 10.1145/2506583.2506635.

[124] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. "Network motifs: Simple building blocks of complex networks". In: *Science* 298.5594 (2002), pp. 824–827. DOI: 10.1126/science.298.5594.824.

[125] Geon Lee, Jihoon Ko, and Kijung Shin. "Hypergraph motifs: Concepts, algorithms, and discoveries". In: *Proceedings of the VLDB Endowment* 13.11 (2020), pp. 2256–2269. DOI: 10.14778/3407790.3407823.

[126] Ron Milo et al. “Superfamilies of evolved and designed networks”. In: *Science* 303.5663 (2004), pp. 1538–1542. DOI: 10.1126/science.1089167.

[127] Quintino Francesco Lotito, Federico Musciotto, Alberto Montresor, and Federico Battiston. “Higher-order motif analysis in hypergraphs”. In: *Communications Physics* 5 (2022), p. 79. DOI: 10.1038/s42005-022-00858-7.

[128] Nicholas W. Landry, Jean-Gabriel Young, and Nicole Eikmeier. “The simpliciality of higher-order networks”. In: *EPJ Data Science* 13 (2024), p. 17. DOI: 10.1140/epjds/s13688-024-00458-1.

[129] Timothy LaRock and Renaud Lambiotte. “Encapsulation structure and dynamics in hypergraphs”. In: *Journal of Physics: Complexity* 4.4 (2023), p. 045007. DOI: 10.1088/2632-072X/ad0b39.

[130] V. Chvátal. “A greedy heuristic for the set-covering problem”. In: *Mathematics of Operations Research* 4.3 (1979), pp. 233–235. DOI: 10.1287/moor.4.3.233.

[131] Thomas Eiter and Georg Gottlob. “Identifying the minimal transversals of a hypergraph and related problems”. In: *SIAM Journal on Computing* 24.6 (1995), pp. 1278–1304. DOI: 10.1137/S0097539793250299.

[132] Michael L. Fredman and Leonid Khachiyan. “On the complexity of dualization of monotone disjunctive normal forms”. In: *Journal of Algorithms* 21.3 (1996), pp. 618–628. DOI: 10.1006/jagm.1996.0062.

[133] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco: W. H. Freeman, 1979.

[134] C. T. Yu and M. Z. Özsoyoğlu. “An algorithm for tree-query membership of a distributed query”. In: *Proceedings of COMPSAC 79, the IEEE Computer Society’s Third International Computer Software and Applications Conference*. 1979, pp. 306–312. DOI: 10.1109/CMPSAC.1979.762509.

[135] Robert E. Tarjan and Mihalis Yannakakis. “Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs”. In: *SIAM Journal on Computing* 13.3 (1984), pp. 566–579. DOI: 10.1137/0213035.

[136] Philip A. Bernstein and Nathan Goodman. “Power of natural semijoins”. In: *SIAM Journal on Computing* 10.4 (1981), pp. 751–771. DOI: 10.1137/0210059.

[137] David Maier. *The Theory of Relational Databases*. Rockville, MD: Computer Science Press, 1983.

[138] Catriel Beeri, Ronald Fagin, David Maier, and Mihalis Yannakakis. “On the desirability of acyclic database schemes”. In: *Journal of the ACM* 30.3 (1983), pp. 479–513. DOI: 10.1145/2402.322389.

[139] Mihalis Yannakakis. “Algorithms for acyclic database schemes”. In: *Proceedings of the 7th International Conference on Very Large Data Bases (VLDB)*. 1981, pp. 82–94.

[140] Ronald Fagin. “Degrees of acyclicity for hypergraphs and relational database schemes”. In: *Journal of the ACM* 30.3 (1983), pp. 514–550. DOI: 10.1145/2402.322390.

[141] Steffen Klamt, Utz-Uwe Haus, and Fabian Theis. “Hypergraphs and cellular networks”. In: *PLoS Computational Biology* 5.5 (2009), e1000385. DOI: 10.1371/journal.pcbi.1000385.

[142] Awad H. Al-Mohy and Nicholas J. Higham. "Computing the action of the matrix exponential, with an application to exponential integrators". In: *SIAM Journal on Scientific Computing* 33.2 (2011), pp. 488–511. DOI: 10.1137/100788860.

[143] Leonie Neuhäuser, Andrew Mellor, and Renaud Lambiotte. "Multibody interactions and nonlinear consensus dynamics on networked systems". In: *Physical Review E* 101 (2020), p. 032310. DOI: 10.1103/PhysRevE.101.032310.

[144] Guilherme Ferraz de Arruda, Giovanni Petri, and Yamir Moreno. "Social contagion models on hypergraphs". In: *Physical Review Research* 2 (2020), p. 023032. DOI: 10.1103/PhysRevResearch.2.023032.

[145] Iacopo Iacopini, Giovanni Petri, Alain Barrat, and Vito Latora. "Simplicial models of social contagion". In: *Nature Communications* 10 (2019), p. 2485. DOI: 10.1038/s41467-019-10431-6.

[146] Dengyong Zhou, Olivier Bousquet, Thomas Navin Lal, Jason Weston, and Bernhard Schölkopf. "Learning with local and global consistency". In: *Advances in Neural Information Processing Systems 16*. MIT Press, 2004, pp. 321–328.

[147] Xiaojin Zhu, Zoubin Ghahramani, and John Lafferty. "Semi-supervised learning using Gaussian fields and harmonic functions". In: *Proceedings of the 20th International Conference on Machine Learning (ICML)*. 2003, pp. 912–919.

[148] Yifan Feng, Haoxuan You, Zizhao Zhang, Rongrong Ji, and Yue Gao. "Hypergraph neural networks". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 3558–3565. DOI: 10.1609/aaai.v33i01.33013558.

[149] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. "Simplifying graph convolutional networks". In: *Proceedings of the 36th International Conference on Machine Learning*. Vol. 97. Proceedings of Machine Learning Research. 2019, pp. 6861–6871.

[150] Sunwoo Kim, Soo Yong Lee, Yue Gao, Alessia Antelmi, Mirko Polato, and Kijung Shin. "A survey on hypergraph neural networks: An in-depth and step-by-step guide". In: *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '24)*. 2024. DOI: 10.1145/3637528.3671457.

[151] Alessia Antelmi, Gennaro Cordasco, Mirko Polato, Vittorio Scarano, Carmine Spagnuolo, and Dingqi Yang. "A survey on hypergraph representation learning". In: *ACM Computing Surveys* 56.1 (2023), p. 24. DOI: 10.1145/3605776.

[152] Austin R. Benson, Rediet Abebe, Michael T. Schaub, Ali Jadbabaie, and Jon Kleinberg. "Simplicial closure and higher-order link prediction". In: *Proceedings of the National Academy of Sciences* 115.48 (2018), E11221–E11230. DOI: 10.1073/pnas.1800683115.

[153] Naganand Yadati, Vikram Nitin, Madhav Nimishakavi, Prateek Yadav, Anand Louis, and Partha Talukdar. "NHP: Neural hypergraph link prediction". In: *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM)*. 2020, pp. 1705–1714. DOI: 10.1145/3340531.3411870.

[154] Lada A. Adamic and Eytan Adar. "Friends and neighbors on the Web". In: *Social Networks* 25.3 (2003), pp. 211–230. DOI: 10.1016/S0378-8733(03)00009-1.

[155] Giulia Cencetti, Federico Battiston, Bruno Lepri, and Márton Karsai. “Temporal properties of higher-order interactions in social networks”. In: *Scientific Reports* 11 (2021), p. 7028. DOI: 10.1038/s41598-021-86469-8.

Index

acyclic hypergraph, 103

barycentric seriation, 25
biclustering, 26

clique expansion, 12
co-clustering, 26
conformal hypergraph, 13
    test, 13

directed hypergraph, 11
dual hypergraph, 11

Eclat, 83

Fiduccia–Mattheyses, 74
frequent itemsets, 83
fuzzy hypergraph, 11

h-motif, 94
hitting set, 102
hyper-core, 62
hypergraph, 10
hypergraph Louvain, 72
hypergraph modularity, 71
hypergraph partitioning, 74
`hypergraph_louvain`, 72

incidence matrix, 11
intrusion, 37

Jaccard similarity
    weighted, 48

`layout_membership`, 51

membership-aware layout, 50
    algorithm, 51
multidimensional scaling, 49

quality function deployment, 105

s-betweenness, 60
s-closeness, 60
s-line graph, 12
s-walk, 58
seriation, 23
    barycentric, 25
set cover, 102
simpliciality, 94
star expansion, 12

threshold contagion, 115
transitive closure, 49



From group interactions to decisions

Teams, production steps, chemical reactions, co-authored papers, and news articles connect more than two entities at once. Hypergraphs represent these group interactions without breaking them into pairs. This book shows how to represent, draw, analyze, and use them in Python.

It covers incidence-matrix seriation, PAOH and UpSet views, hull, bubble, polygon, and similarity-based drawings with linked primal and dual views, interchange through the Hypergraph Interchange Format, s-walk and eigenvector centralities, hypergraph modularity, spectral clustering and partitioning, frequent patterns and validated hyperedges, higher-order motifs, simpliciality and null models, set cover, transversals, acyclicity and directed hypergraphs, quality function deployment, diffusion and contagion, and learning on hypergraphs.

Every method is implemented in `hatvis`, an open-source Python package and the successor of the HAT-VIS toolbox for MATLAB and of the HyperGodot viewer. Each figure in the book is produced by an example script of the package; the same scripts run as Jupyter notebooks.

```
pip install hatvis
```

github.com/ComplexSystemsIntelligence/hatvis