UCL Université catholique de Louvain

École polytechnique de Louvain (EPL)

epl

Finding informative cuboids in graphs with numerical attributes

Dissertation presented by
Alex MATTENET

for obtaining the Master's degree in
Computer Science and Engineering

Supervisor
Siegfried NIJSSEN

Readers
Amine GHRAB, John AOGA

Academic year 2017-2018

Abstract

Graphs are an indispensable model to represent networked data. As our world becomes more interconnected, it is increasingly important to have the tools to analyze and understand large, complicated graphs. This is why the topic of graph mining is seeing rising interest. Moreover, the availability of richer graph dataset — containing not only the network structure but also attributes on the edges and vertices — have created the need for new graph mining frameworks.

One such framework is an adaptation of OLAP (On Line Analytical Processing) for networked data, called Graph Cube. In the graph cube, the network is aggregated into a smaller graph which summarizes the connections between nodes sharing some properties. For example in a social network, instead of edges between persons, the summarized graph will show groups of people with a shared property, such as nationality or common interest.

In this thesis, I explore one of the open questions in the Graph Cube Mining framework. Namely, how to generate informative cuboids, when some of the attributes are numerical. I present a new measure of informativeness for cuboids, based on a yet unexplored connection between Graph Cubes and Stochastic Blockmodels. Then, I explore three different approaches to build informative cuboids with numerical attribute: A binning approach, an approach based on Expectation Maximization, and an approach based on clustering.

Acknowledgments

I would first and foremost like to thank my thesis supervisor Prof. Siegfried Nijssen for his patience and continued guidance, feedback and support throughout my work on this thesis.

I would also like to sincerely thank Amine Ghrab and Florian Demesmaeker from Eura Nova for their time and constructive comments. Thank you also to Eura Nova in general. It was fantastic to have the opportunity to collaborate with this company, and I enjoyed the warm and hard-working environment.

Last but not least, I am grateful to [redacted], who helped me stay sane in the final stages of the redaction of this thesis, as well as Elias and my family, who consistently provided encouragements and moral support throughout my academic studies.

A note of the layout of this thesis

The layout of this thesis tries to follow the design guidelines of Edward Tufte [1]. Lines are kept short to improve the comfort of reading. Footnotes, comments and small figures are displayed in the margin, next to the text*, in order not to disturb the flow of reading.

* like this for example

References are cited in the IEEE style, with a number in brackets. When a new reference is introduced, the title, author and year of publication are displayed in the margin, like this [1]. The complete bibliographical references are found at the end of this work, under the Bibliography section.

[1]: *Beautiful evidence*, Tufte, 2006

In the PDF version of this text, cross-references for figures, sections, citations, tables, and URLs are clickable links, which bring you to the corresponding part of the document.

Contents

1
Introduction

As our world becomes more interconnected and interdepedent, graph-structured data and its analysis takes a more important role. The emergence of large and complex graphs in topics ranging from social science to bioinformatics has lead to an increased interest in the topic of graph mining. Moreover, the availability of richer graph dataset — containing not only the network structure but also attributes on the edges and vertices — have created the need for new graph mining frameworks.

One such framework is an adaptation of OLAP (On Line Analytical Processing) for networked data, called Graph Cube [2] [3]. In the graph cube, the network is aggregated into a smaller graph which summarizes the connections between nodes sharing some properties. For example in a social network, instead of edges between persons, the summarized graph will show groups of people with a shared property, such as nationality or common interest. Aggregate measures can be calculated for nodes, showing for example the average age of people, as well as for edges — e.g. showing the average duration of the link.

[2]: "Graph cube", Zhao, Li, Xin, *et al.*, 2011

[3]: "A Framework for Building OLAP Cubes on Graphs", Ghrab, Romero, Skhiri, *et al.*, 2015

In his Masters' Thesis [4], Florian Demesmaeker presented a technique to find interesting patterns in large graph cubes, later published in a paper [5]. In this context, patterns represent repeated associations between two nodes according to some of their features. For example, in a social network, the relationships between Belgian and Chinese people is one such pattern.

[4]: "Graph cube mining", Demesmaeker, 2017

[5]: "Discovering interesting patterns in large graph cubes", Demesmaeker, Ghrab, Nijssen, *et al.*, 2017

The pattern mining framework presented in [4] and [5] has the limitation that handles only categorical attributes — e.g. nationality or gender — and not numerical attributes, like age, revenue, number of followers, etc. In this work, we explore different ways of overcoming this limitation.

This text is organised as follows:

- In chapter 2, I present the graph cube model (section 2.1), as well as the pattern mining framework developped by Demesmaeker (section 2.2). Then, I explain why this framework is ill-suited for numerical attributes, and make the case that what we need to include numerical attributes is a quality measure for cuboids (section 2.3).
- In chapter 3, I propose a quality measure for cuboids based on the Minimum Description Length principle. This quality measure is based on the theory of stochastic blockmodels. I start by presenting stochastic blockmodels in section 3.1. Then, I compare systematically the stochastic blockmodel framework with the graph cube mining framework (section 3.2).

Finally, I define the quality measure in section 3.3, and explain how it can be used theoretically to handle numerical attributes.

- In chapter 4, I explore a first approach for building cuboids with numerical attributes, based on the idea of binning. I propose a greedy algorithm in section 4.2. Then, I propose an experimental setup to evaluate its performance on synthetic datasets (section 4.3).
- In chapter 5, I explore a second approach for building cuboids with numerial attributes based on Expectation Maximization (EM). I propose a latent variable model for multidimensionnal networks (section 5.1), and build an EM algorithm to estimate the maximum likelihood parameters for this model (section 5.3). Finally, I discuss the usefulness of this approach in section 5.4.
- In chapter 6, I present a third approach for building cuboids with numerical attributes, based on clustering. I present a description of vertices based on their connective behavior (section 6.1) along with a corresponding distance metric (section 6.2). Then, I use these in conjunction with a k-medoids algorithm to build interesting cuboids in section 6.3. Finally, I give some preliminary results of this approach in section 6.4.
- I conclude this thesis in chapter 7 and discuss further work.

2

Previous work: Graph cube mining and the problem with numerical attributes

This thesis expands on the work introduced in Florian Demesmaeker's thesis last year [4]. Its topic was the search for interesting patterns in graph cubes. All the necessary background will be presented in this section.

In section 2.1, I present the graph cube model as described by Zhao, Li, Xin, *et al.* [2]. I start by presenting the (regular) data cube model, and its extension to multidimensional graphs.

Then, in section 2.2, I present the setting studied by Demesmaeker [4] for the search of interesting patterns in graph cubes. The notions of pattern, support, expected support and interestingness are defined. Finally, I show the algorithm developped by Demesmaeker [4].

Finally, in section 2.3, I explain the difference between categorical and numerical attributes, and present the core problem behind this thesis: how can we adapt the work of [4] to handle numerical attributes?

[4]: "Graph cube mining", Demesmaeker, 2017

[2]: "Graph cube", Zhao, Li, Xin, *et al.*, 2011

2.1 The graph cube model

2.1.1 OLAP data cube

On-Line Analytical Processing (OLAP) is a technique for performing complex analysis over the information stored in data warehouses [6]. With OLAP, analysis is based on the calculation of a large number of aggregates by grouping over multiple dimensions. The typical model used to conceptualize the data for an OLAP setting is the data cube [7]. The data cube is a multidimensional array of *measures* or values, based on a set of *dimensions*. For example, for an application in retail sales, the dimensions will include `product`, `customer`, `location`, and `time`. If the measure of interest is total sales, then a *cell* in the data cube represents the aggregate number of sales for a specific aggregation of `product`, `costumer`, `location`, and `time`. For example, the blue cell in Figure 2.1 holds the total number of sales for product P1, for all customers in France in 2017.

Dimensions in OLAP are often hierarchical. In our example, the dimension `time` can be divided into `year` > `quarter` > `month` > `day`; the dimension `location` can be divided into `region` > `country` > `city`, and so on. These all give us a different granularity of the aggregations on the data cube: `year` corresponds to a high level of granularity, whereas `day` corresponds to

[6]: "The cube data model: a conceptual model and algebra for on-line analytical processing in data warehouses", Datta and Thomas, 1999

[7]: "An Overview of Data Warehousing and OLAP Technology", Chaudhuri and Dayal, 1997

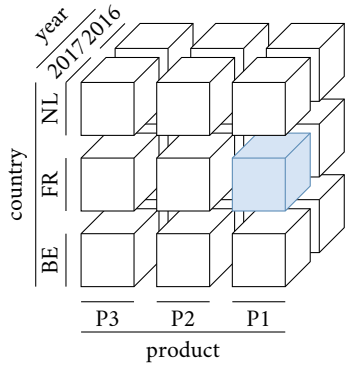


Figure 2.1: Aggregation on `country`, `product` and `year` in the data cube for a retail application.

a low level of granularity.

Two operations can be used to move between aggregations of the data cube: *roll-up* aggregates the current cube more by removing dimensions, and *drill-down* aggregates less by adding one dimension.

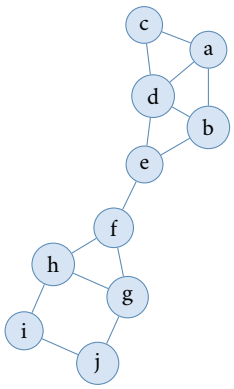
Other common operations in OLAP are *slice* (selecting some dimensions of the cube), *dice* (selecting a particular value on one of the dimensions, e.g. only looking at product P1), and *pivot*. Pivot transform the measure in the current aggregation into an additional dimension in a new cube. For example, the result of a pivot on a two-dimensional cube showing total sales by product and by year would be a three dimensional cube having as attributes "product", "year" and "total sales", which allows to make new measures with respect to this new attribute — "total sales".

2.1.2 Extension to multidimensional graphs

When dealing with networked datasets, such as data from social networks, the value and interest lies in the structure of the network itself — that is, in who is connected to who. Using a traditional OLAP framework which doesn't account for the network structure of the data will miss the most insightful part and will not be help the analysis sufficiently.

In [2], Zhao, Li, Xin, *et al.* introduce an extension of the data cube framework to multidimensional networks called *Graph Cube*. This extension supports all the traditional OLAP queries, and adds an additional graph structure to the data cube. This graph structure, which is derived from the structure of the network being analyzed, enriches the results and allows for more insightful analysis of the dataset.

Multidimensional networks Let us consider the toy example in Figure 2.2. It presents a small social network consisting of several individuals interconnected with friend relationships. It has ten vertices (labelled a to j) and 14 edges. Every vertex in the network has a tuple of values for the attributes country, language, and number of posts made by that person. The graph structure along with the tuples for all vertices forms the *multidimensional network*.



Vertex	Country	Language	Posts
a	BE	en	280
b	NL	en	210
c	FR	fr	320
d	BE	en	250
e	NL	nl	200
f	BE	nl	110
g	BE	fr	130
h	NL	en	140
i	FR	fr	250
j	FR	fr	300

Figure 2.2: Toy example of a multidimensional network with its graph and multidimensional vertex attribute table.

We will now give the formal definition of these concepts, taken from [2]:

Definition 1. *A **multidimensional network** $\mathcal{N}$ is a graph denoted as $\mathcal{N} = (V, E, A)$, where V is a set of vertices, $E \subseteq V \times V$ is a set of edges* and $A =$*

* This definition of graph allows for directed edges as well as undirected edges. A tuple (a, b) in V represents an edge *from a to b*; and an undirected edge will be represented by two edges, one for each direction. An undirected graph will simply have the property $\forall (a, b) \in V : (b, a) \in V$.

$\{A_1, A_2, \ldots, A_n\}$ is a set of n vertex-specific attributes, i.e. $\forall u \in V$, there is a multidimensional tuple $A(u) = (A_1(u), A_2(u), \ldots, A_n(u))$, where $A_i(u)$ is the value of the i-th attribute of the vertex u, with $1 \leq i \leq n$. A is called the dimension of the network $\mathcal{N}$.

The formal representation of our toy example in Figure 2.2 is (V, E, A) with the set of vertices $V = \{a, \ldots j\}$, the set of edges $E = \{(a, b), (a, c), (a, d), (b, a), \ldots, (j, i)\}$ and the set of attributes as reported in the table of Figure 2.2, e.g. $A(b) = (A_1(b), A_2(b), A_3(b)) = (\text{NL}, \text{en}, 2010)$

Aggregate networks By analogy to the data cube aggregations, we define the possible aggregations upon multidimensional networks. Given a network with n attributes, there exist 2^n multidimensional spaces (aggregations). However the measures for each aggregation are no longer arrays of numerical values, but *aggregate networks*.

This *aggregate network* is constructed in the following way:

1. every vertex in the network is put into the corresponding cell in the aggregated view, as seen in Figure 2.3;
2. vertices in the same cell are grouped together into an *aggregated vertex*;
3. aggregated vertices get a weight (= the value in the cell for traditional OLAP);
4. grouped edges get a weight (here, the number of edges between grouped vertices);
5. self-loops occur when there are edges inside a cell.

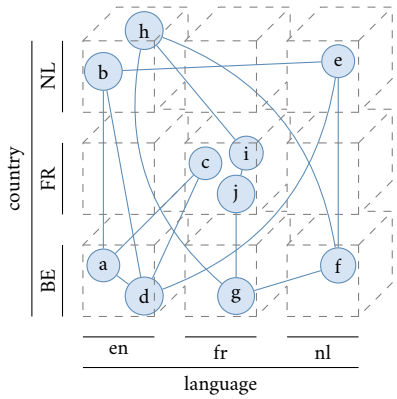


Figure 2.3: The toy network with its vertices grouped by the aggregation (country, language, *)

Figure 2.4 and 2.5 present the aggregate network for the aggregation (country, language, *) and (country, *, *) respectively. As we can see, the condensed vertex for (BE, en, *) has a weight of two, because it is an aggregate view of two vertices from the toy network: a and d. This vertex has a self-loop with weight 1 because there is 1 edge between the vertices it aggregates (a–d). In the same way, the edge (BE, *, *) – (FR, *, *) in Figure 2.5 has weight 3 because it is the aggregation of three edges between Belgian and French individuals: a–c, c–d and g–j.

We chose the COUNT(·) as the measure to derive the weights for aggregated vertices and edges, but more complicated functions can also be chosen. For example, we could measure the average age of an aggregated vertex, or the average degree. In the rest of this work, as in [2], we will chose COUNT(·) as the default aggregate function for all aggregations, but the algorithms can easily be generalized to accommodate other aggregate functions.

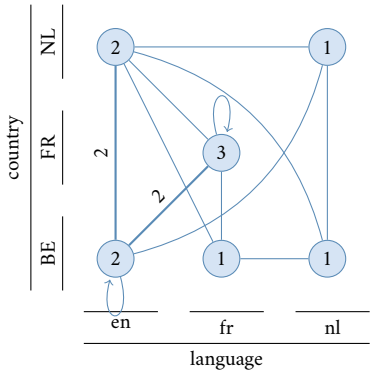


Figure 2.4: Aggregate network w.r.t. the aggregation (country, language, *) of the toy network. Edges with no label have weight 1.

The formal definition of aggregate networks is as follows:

Definition 2. *Given a multidimensional network $\mathcal{N} = (V, E, A)$ and a possible aggregation $A' = (A'_1, A'_2, \ldots, A'_n)$ of A where A'_i equals A_i or $*$, the **aggregate network w.r.t.** A' is a weighted graph $G' = (V', E', W_{V'}, W_{E'})$ where*

1. *$\forall[v]$, a nonempty equivalence class of V where $[v] = \{u \in V \mid \forall A_i \in A : A'_i(u) = A'_i(v)\}$, $\exists v' \in V'$ as representative of $[v]$. The weight of v',*

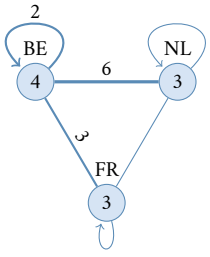


Figure 2.5: Aggregate network w.r.t. the aggregation (country, *, *) of the toy network. Edges with no label have weight 1.

[2]: "Graph cube", Zhao, Li, Xin, *et al.*, 2011

$w(v') = \Gamma_V([v])$, *where* $\Gamma_V(\cdot)$ *is an aggregate function defined upon vertices.* v' *is therefore called a* ***condensed vertex*** *or* ***aggregated vertex****;*

2. $\forall u', v' \in V'$ *and a nonempty edge set* $E_{u',v'} = \{(u,v) | u \in [u]$ *represented as* $u', v \in [v]$ *represented as* $v', (u,v) \in E\}$, $\exists e' \in E'$ *as a representative of* $E_{(u',v')}$*. The weight of* e', $w(e') = \Gamma_E(E_{(u',v')})$, *where* $\Gamma_E(\cdot)$ *is an aggregate function defined upon edges.* e' *is therefore called a* ***condensed edge*** *or* ***aggregated edge****.*

For brevity, we will omit the $*$ from the description of aggregations; e.g. the aggregation (country, *, *) will be designed as (country), the aggregation (*, language, posts) as (language, posts) and the aggregation (*, *, *) as (*).

Graph cube lattice The *graph cube* is comprised of all possible aggregate networks over the multidimensional network $\mathcal{N} = (V, E, A)$. These different aggregate networks can be shown to form a lattice structure, called the *graph cube lattice*, which is of particular interest for us. The following paragraphs will present this lattice structure.

First let's define some more vocabulary. An aggregation A' (as defined in Definition 2) along with its aggregate network is also called a **cuboid**[†]. Following the same definition, the **size** of a cuboid is its number of condensed edges and vertices $|V'| + |E'|$. For a cuboid A', $\dim(A')$ is defined as the set of non-$*$ dimensions of A'. For example, if $A' = (\text{country}, *, \text{posts})$, $\dim(A') = \{\text{country}, \text{posts}\}$.

Consider two cuboids A' and A''. A' is an **ancestor** of A'' if $\dim(A') \subseteq \dim(A'')$, and conversely A'' is a **descendent** of A'. Specifically, if $|\dim(A'')| = |\dim(A')| + 1$, then A' is a **parent** of A'' and A'' is a **child** of A'. If $|\dim(A')| = |\dim(A'')| = l$, then A' and A'' are **siblings** at the l level of the graph cube.

There are two distinguished cuboids in the cube. The first is the **base cuboid** which is the descendant of all other cuboids, and has $|\dim(A_{\text{base}})| = n$. The other is the **apex cuboid** $A_{\text{apex}} = (*, *, \ldots, *)$. The apex is an ancestor of all other cuboids.

If we denote the set of all cuboids of a graph as 2^A, i.e. the power set of A, then we can define the **graph cube lattice** as that set equipped with the partial ordering $\subseteq$: $\mathcal{L} = \langle 2^A, \subseteq \rangle$. The full lattice for the toy example is shown in Figure 2.1.2.

Since the apex cuboid has no dimensions, it aggregates all the vertices into a single condensed vertex, with an optional self-loop. The weight of that condensed vertex will be the total number of vertices of the multidimensional network, $|V|$, and the weight of the self-loop will be the total number of edges $|E|$. An illustration of this for the toy example is shown in Figure **??**.

As we go down in the lattice, the views get more and more granular, until the base cuboid which is the most descriptive in the graph cube lattice, since it has the most condensed vertices and edges.

[†] We will use the terms aggregation, cuboid, view and query interchangeably throughout the document.

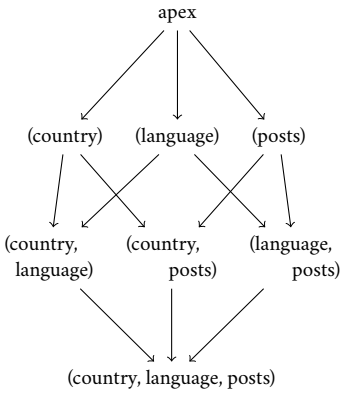


Figure 2.6: Graph cube lattice $\mathcal{L} = \langle 2^A, \subseteq \rangle$ for the toy example. Each vertex is a cuboid. Edges represent the ancestor relationship, i.e. $A \to B$ is $A \subseteq B$.

Figure 2.7: Apex cuboid for the toy example. All vertices have been condensed into a single vertex whose weight is equal to the number of vertices $|V|$. All edges have been condensed into a single edge whose weight is equal the number of edges $|E|$

2.2 Mining patterns in a graph cube

We are interested in finding surprising patterns in multidimensional networks. For this, we take as our starting point the work presented by Florian

Demesmaeker in his master thesis last year [4], and later the same year in a paper [5]. In his work, we are given 1. a definition of pattern, 2. a measure of interestingness for patterns, and 3. an algorithm to search efficiently for the most interesting patterns. These will be presented in this section.

[5]: "Discovering interesting patterns in large graph cubes", Demesmaeker, Ghrab, Nijssen, *et al.*, 2017

In this context, a pattern is an edge of any aggregated graph in the graph cube lattice. The problem statement is as follows: We are given a multidimensional network $\mathcal{N} = (V, E, A)$, and its graph cube lattice, containing $2^{|A|}$ cuboids and as many corresponding graphs. We are looking for the aggregate edges in any of these graphs that are surprising with respect to some prior knowledge we have on the data.

2.2.1 Pattern definitions

We start by formalizing the definition of a pattern. We work with multidimensional networks, so what we observe are edges between vertices. Every vertex has some specific values for all the defined attributes. What we call a pattern is the (repeated) appearance of an edge between vertices fitting some description T_1 and vertices fitting some description T_2. Formally, we have the following:

Definition 3. *In a network $\mathcal{N} = (V, E, A)$, a **pattern** P is defined by a selection of attributes $S \subseteq A$ and two tuples of values T_1 and T_2 defined over the attributes in S. The pattern P is the edges (u, v) such that $S(u) = T_1, S(v) = T_2$ with $u, v \in V$.*

Every pattern corresponds exactly to an edge in one of the aggregated graphs of the lattice: for any selection of attributes S, the pattern given by the tuples of values T_1, T_2 corresponds to an edge in the cuboid with regards to the aggregation S. In this cuboid, we can find a vertex u' representative of $[u] = \{u \in V \mid S(u) = T_1\}$ and a vertex v' representative of $[v] = \{v \in V \mid S(u) = T_2\}$. This vertex corresponds uniquely to the pattern P.

An example of a pattern is given in Figure 2.8. The pattern P corresponds to friendships between Belgians and Dutchmen. It is formally determined by $S = \{country\}$, $T_1 = (\text{BE})$ and $T_2 = (\text{NL})$, following the notation of Definition 3. We see first the edge corresponding to P in the aggregate network for (country), then every edge in the base graph matching P.

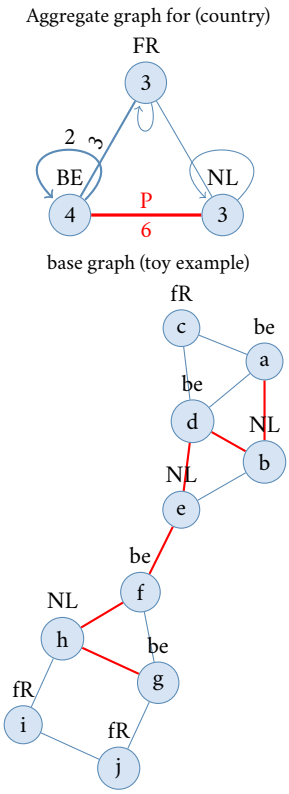


Figure 2.8: Example of a pattern P in the toy example. P is given by the tuples $T_1 = (\text{BE})$ and $T_2 = (\text{NL})$ in the selection of attributes $S = \{\text{country}\}$. The corresponding edge is shown in red in the (country) cuboid, and all the edges matching P are shown in red in the base graph.

2.2.2 Support

We can see in Figure 2.8 that there are 6 occurrences of friendships between Belgians and Dutchmen in our toy example — these are the 6 edges in red. This is also reflected in the weight of the edge P in the aggregated graph for (country). This value is called the *support* of P. Formally, we state

Definition 4. *The **support** $\text{supp}_S(T_1, T_2)$ of a pattern $P = (T_1, T_2)$ in a cuboid S is defined as the number of occurrences of P. That is the number of edges in the multidimensional network $\mathcal{N} = (V, E, A)$ respecting the conditions of P.*

$$\text{supp}_S(T_1, T_2) = \sum_{(u,v) \in E} \delta(u, v)$$

where δ is an indicator function equal to 1 for edges matching P and 0 otherwise:

$$\delta(u,v) = \begin{cases} 1 & \text{if } S(u) = T_1 \wedge S(v) = T_2 \\ 0 & \text{otherwise} \end{cases}$$

This definition works for directed graphs. For undirected graphs, the edges will be counted twice — once as (u,v) and once as (v,u). In that case, the actual support is given by $\sum_{(u,v)\in E} \delta(u,v)/2$[‡].

The higher the support of P, the more it appears in the base graph. Conversely, the lower the support, the less it appears in the base graph. A support of 0 means that no edge matching P can be found in the base graph. Patterns with a support of 0 will thus not appear as edges in a cuboid of the lattice, since they have no representative in the base graph.

[‡] In Demesmaeker [4], which only deals with undirected graphs, this is equivalently defined as $\sum_{(u,v)\in E} \delta(u,v)$ with

$$\delta(u,v) = \begin{cases} 1 & \text{if } S(u) = T_1 \wedge S(v) = T_2 \\ & \vee S(u) = T_2 \wedge S(v) = T_1 \\ 0 & \text{otherwise} \end{cases}$$

2.2.3 *Expected support*

Given some model of our data and some prior knowledge, we can calculate expectations for the support of a given pattern P. This will be the *expected support* of P (given the prior knowledge S'). This expected support will later be useful to quantify the interestingness of P.

The model we chose here is inspired by the partition model [8], and adapted to work with graph cubes. In the original binary itemset mining setting, this model works in the following way: if we are given an itemset X, consider a partition $\mathcal{P} = P_1, \ldots, P_M$ of X with $\bigcup_{i=1}^{M} P_i = X$, and $P_i \cap P_j = \emptyset$ for $i \neq j$. Under this model, we expect the partitions to be independent from each other, so the expected frequency[§] of X is the product of the frequencies of its parts, i.e. $\prod_{i=1}^{M} \text{freq}(P_i)$.

[8]: "Interesting Patterns", Vreeken and Tatti, 2014

[§] the frequency is simply the ratio between the support and the total number of transactions in the database.

For n items, there are 2^n possible partitions, and therefore we can compute 2^n expected frequency measures for an every itemset.

Our definition of expected support will be based on this idea. For a pattern P defined over the aggregation S, we express the support of P with respect to a more aggregated cuboid $S' \subset S$. Similarly to the itemset setting, there are $2^{|S|} - 1$ cuboids $S' \subset S$ to chose from, yielding as many expected supports for P.

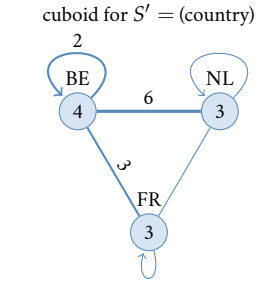
First we define the probability p of a vertex having certain attributes defined by a tuple T given a more aggregated view of the network: The probability $p_{S,S'}(T)$ of a vertex having values T over attributes S is the fraction of vertices having values T among those having values T', where T' is defined over S'. p is always between 0 and 1, and can be seen as the probability $\Pr(T \mid T')$.

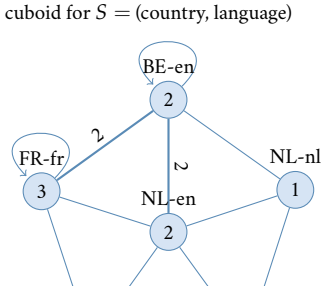
Definition 5. *The **proportion** $p_{S,S'}(T)$ of a tuple of values T defined over a set of attributes S with respect to a set of attributes $S' \subset S$ is given by*

$$p_{S,S'}(T) = \frac{\sum_{u\in V} \gamma_S(u)}{\sum_{u\in V} \gamma_{S'}(u)}$$

where $\gamma_S(u)$ is an indicator function equal to 1 when u matches T for the attributes in S:

$$\gamma_S(u) = \begin{cases} 1 & \text{if } S(u) = T \\ 0 & \text{otherwise} \end{cases}$$





For example, let us calculate the proportion of french-speaking Belgians among all Belgians. Here $S = \text{country, language}$, $S' = \text{country}$, $T = (\text{BE, fr})$. The relevant cuboids are shown in Figure 2.9. We can see in the weight of the BE vertex that there are 4 Belgians in total in the database, and in the weight of the BE-fr vertex that only one french-speaking Belgian. Following Definition 5, we have $p_{S,S'}(T) = \frac{\text{weight(BE-fr)}}{\text{weight(BE)}} = \frac{1}{4}$.

The expected support of a pattern $P = (T_1, T_2)$ corresponding to a cuboid S can be computed from the more aggregated cuboid $S' \subset S$. Let $P' = (T'_1, T'_2)$ be P restricted to S', i.e. P from which we remove the values of attributes $S \setminus S'$. We will compute $\Pr(P \mid P')$ the probability of observing P given the fact that we observe P'. This probability is calculated under the assumption that the data is generated from a partition model: It is assumed that each vertex has the same probability to be seen. As a consequence, the probability of observing an edge $P = (T_1, T_2)$ is the product of the proportions of its two end points T_1 and T_2 with respect to S'. If we sample a random edge from G, the probability of the endpoint attributes matching those of P given P' is given by

$$\Pr(P \mid P') = p_{S,S'}(T_1)p_{S,S'}(T_2) \tag{2.1}$$

This is the probability that one of the $\text{supp}(P')$ edges matches P. The total expected support can thus be seen as $\Pr(P \mid P')$ multiplied by the support of P', by the independence assumption.

Figure 2.10: Apex cuboid for the toy example.

However, this is not always correct for undirected graphs. Consider the (country) network given in Figure 2.9. Its only ancestor is the apex that contains a single node and a self-loop (Figure 2.2.3). Here, $p_{S,S'}(\text{BE}) = 0.4, p_{S,S'}(\text{NL}) = 0.3, p_{S,S'}(\text{FR}) = 0.3$. It is clear that the sum of these proportions must be 1. We also know that, for any of the 14 edges in the apex graph, the sum of the probabilities for all combination of end points must be one:

$$\sum_{u,v \in \{\text{BE, FR, NL}\}} \Pr((u, v) \mid (*, *)) = 1$$

However, if we just apply Equation 2.1, we run into a problem. In order for the sum to equal to one, it needs to include both $\Pr((\text{BE, FR}) \mid (*, *))$ and $\Pr((\text{FR, BE}) \mid (*, *))$. However these two probabilities correspond to the same (undirected) edge BE-FR, so this edge has the probability $\Pr((\text{BE, FR}) \mid (*, *)) + \Pr((\text{FR, BE}) \mid (*, *)) = 2p_{S,S'}(\text{FR})p_{S,S'}(\text{BE})$.

In general, this situation occurs when drawing a non self-loop edge from a self-loop in the ancestor cuboid. This induces a bias towards non self-loops being more likely than self-loops for undirected graphs.

Definition 6. *The **Expected support** $E[\text{supp}_S(T_1, T_2)]$ of a pattern (T_1, T_2) defined over a cuboid $S \subseteq A$ is derived from a more aggregated selection of attributes $S' \subset S$. It is given by*

$$E[\text{supp}_S(T_1, T_2)] = \text{supp}_{S'}(T_1, T_2)p_{S,S'}(T_1)p_{S,S'}(T_2)$$

Following the same example as above, the expected support of the edge BE-FR is $\text{supp}_{(*)}(\text{BE, FR}) * p_{(\text{country}),(*)}(BE) * p_{(\text{country}),(*)}(FR) * 2$ (times two because it is an undirected non self-loop taken from a self loop). This is $14 * 0.4 * 0.3 * 2 = 3.36$. This value is a good estimate, the observed value being 3.

In the next section, we will see how we can compare the expected and actual support of an edge to determine its interestingness.

2.2.4 *Significance of a pattern*

To determine if a pattern P defined over cuboid S is interesting or not, we can compare the expected and actual support of P. Since the expected support is based on the knowledge of an ancestor cuboid $S' \subset S$, we can choose several cuboids S'. In order not to consider an exponential number of ancestor cuboids, it is chosen to restrict the search to the direct ancestors of S, i.e. those just one level above in the lattice, having $|S'| = |S| - 1$. We can compute the expected support $E[\text{supp}_S(P)]_{S'}$ for every one of these direct ancestors S', leading to $|S'| = |S| - 1$ different measures to compare with the actual support of P.

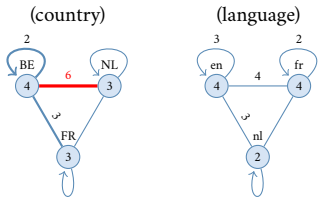
In [4], three different interestingness measures are proposed (absolute difference, ratio and statistical). We will only present and use the last one in this thesis, as it is more well-founded.

[4]: "Graph cube mining", Demesmaeker, 2017

We will take as interestingness measure the probability that a pattern P in cuboid S has indeed the observed support $\text{supp}(P)$ given that P is drawn from its ancestor edge in an ancestor S'. Indeed, the edge P in the cuboid S has a unique matching edge in the cuboid for S' — this is the edge whose end points have the same properties as the end points of P, except for the missing attributes. Several edges in S can have the same ancestor edge in S', and the support of the ancestor edge will always be greater or equal to the support P.

Let p be the probability that one of the edges P' in the ancestor cuboid S' is an edge matching P, as computed in Equation 2.1. Since the edges P' are distributed in S, we can model the probability of observing P given P' by a multinomial distribution. For example, considering $P' = \text{BE-NL}$ (highlighted in red in Figure 2.2.4). It is distributed into 5 edges in the cuboid (country, language). Each edge has its own probability to be drawn from the 9 edges of the cuboid (country). For instance, the probability that the edge $P_1 = ((\text{BE}, \text{en}), (\text{NL}, \text{nl}))$ is drawn from the 9 edges is given by $\frac{\text{weight(BE-en)}}{\text{weight(BE)}} * \frac{\text{weight(NL-nl)}}{\text{weight(NL)}} = \frac{2}{4} * \frac{1}{3} = \frac{1}{6}$.

Ancestor cuboids of (country, language):



cuboid for (country, language)

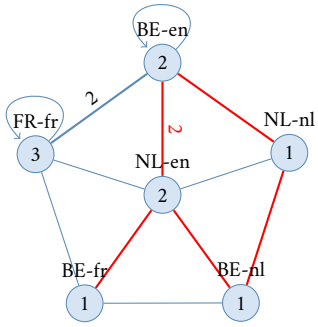


Figure 2.11: (country, language) cuboid for the toy network with its ancestors. The edges from Belgians to Dutchmen are shown in red.

We see an edge in S as a random variable with a probability of success or probability to be present p with respect to an ancestor S'. Then, the probability of observing the actual support can be modeled with the multinomial distribution defined over all the possible edges in S than can be matched to a particular edge in S'. Equation 2.2 defines the probability to observe $\text{supp}(P)$ in a cuboid S with respect to an ancestor $S' \subset S$, where the multidimensional network corresponding to S' is noted as $G' = (V', E', A')$.

$$\Pr(\text{supp}(P)) = \binom{|E'|}{\text{supp}(P)} p^{\text{supp}(P)} (1-p)^{|E'| - \text{supp}(P)} \tag{2.2}$$

This equation is read in the following way:

- The binomial coefficient is the number of possibilities to draw the actual number of edges from the number of aggregated edges, i.e. a selection of $\text{supp}(P)$ edges from a set of $|E'|$ edges.
- The number of edges $|E'|$ is equal to $\text{supp}(P')$.

- p is the probability to draw P from P'. There are $\text{supp}(P)$ edges drawn that way.
- $1-p$ is the probability not to draw P from P'. All the other edges are drawn this way, so $|E'| - \text{supp}(P)$ edges.

For example, let's calculate the probability that the edge P =((BE, en), (NL, en)) has indeed support 2, as seen in Figure 2.9. We chose the (language) cuboid as the ancestor cuboid for this calculation. The ancestor edge is P' =((en, en)), which has a support of 3. The probability p that one of the ancestor edges matches P is $2/4 * 2/4 * 2 = 1/2$[¶].

[¶] Here again we draw a non self-loop from a self-loop ancestor, so we multiply by 2 the product of proportions.

$$\begin{aligned}\Pr(\text{supp}(P)=2) &= \binom{|E'|}{\text{supp}(P)} p^{\text{supp}(P)}(1-p)^{|E'|-\text{supp}(P)} \\ &= \binom{3}{2} p^2(1-p)^1 \\ &= 3 * \frac{1}{2}^2 * \frac{1}{2} = \frac{3}{8}\end{aligned}$$

Since there are more than one ancestor cuboid to consider, we choose to define the interestingness as the highest probability to observe the actual support under the independence model. If we denote the set of all ancestors of S by $S_{\text{anc}} = \{X \in 2^A | 1 + |X| = |S|\}$, we define

$$\text{int}(P) = \max_{S' \in S_{\text{anc}}} \Pr(\text{supp}(P)) \tag{2.3}$$

Another way to understand this it to say that we pick the ancestor cuboid which best explains the observed support under the independence model. It it the probability related to this best model which is used as the interestingness measure.

Continuing the previous example, $\text{int}(((BE, en), (NL, en)))$ is the max between the probability calculated from the (language) cuboid and the (country) cuboid. The probability calculates with the (country) cuboid as ancestor is $\binom{6}{2} * \frac{2*2}{4*3}^2 * (1 - \frac{2*2}{4*3})^4 = 15 * \frac{1}{9} * \frac{1/81}{=} \frac{5}{243}$. Thus we get $\text{int}(((BE, en), (NL, en))) = \max(\frac{3}{8}, \frac{5}{243}) = \frac{3}{8}$.

2.2.5 Algorithm

We will now take a look at the search algorithm presented in [4]. Algorithm 1 performs a search for interesting patterns in multidimensional networks, following all the definitions given in this section. It is an exhaustive search through every cuboid. It takes as input the lattice $\mathcal{L}$ and a probability threshold θ. Patterns with a interestingness $\text{int}(P) \leq \theta$ will be considered "interesting" and returned, while the others will be dismissed as "uninteresting". In the latter case, we say that one of the direct ancestors of the network in which P lies *explains* the pattern with respect to θ.

[4]: "Graph cube mining", Demesmaeker, 2017

Remember that lower interestingness score, paradoxically, mean that the pattern is more unexpected, thus more interesting.

As discussed in **??**, since the lattice $\mathcal{L}$ is exponential in size, the time complexity of the algorithm is also exponential. More specifically, for a base network (V, E, A), if we denote by $n = |A|$ the number of dimensions of the base network (the number of different attributes), and by $m = |E|$ the number of edges, the time complexity of the algorithm is $O(mn2^n)$.

Algorithm 1: search for surprising patterns in the graph cube lattice

Input: $\mathcal{L}$ the lattice containing all the aggregate networks built from a given network
Input: θ the aggregate edge probability threshold
Output: The set of surprising aggregate edges in any network of $\mathcal{L}$ with respect to θ
patterns $\leftarrow \emptyset$
for $\mathcal{N} \in \mathcal{L} \setminus$ apex **do**
    $(V, E, A) \leftarrow \mathcal{N}$
    for $e \in E$ **do**
        interesting $\leftarrow$ true
        ancestors $\leftarrow \{(V', E', A') \mid A' \in A, |A'| = |A| - 1\}$
        for $\mathcal{N}' \in$ ancestors **do**
            p $\leftarrow$ probability$(\mathcal{N}, \mathcal{N}', e)$
            if p$\geq \theta$ **then**
                interesting $\leftarrow$ false
                break
        if interesting **then**
            patterns $\leftarrow$ patterns $\cup (e, p)$
return patterns

2.3 *Limitations when facing numerical attributes*

Although the framework just presented works well with categorical attributes, it is inapropriate when dealing with numerical attributes. Finding a suitable way to make up for this shortcoming is the core problem driving this master's thesis. In this section, I first present the distinction between so-called categorical and numerical attributes. Then, I elaborate on my claim that Demesmaeker [4]'s framework is not suited for numerical attributes. The rest of the thesis will be devoted to the exploration of different ways to adapt the framework to work with numerical attributes in a useful and well-founded way.

2.3.1 *Distinction between categorical and numerical attributes*

Implicit in this previous work is the assumption that attributes are categorical. What we mean by this is that all the vertex-specific attributes are assumed to have a small, finite domain, with different values denoting different, unrelated *categories*.

An attribute characterizing the type of product in a store's inventory is a good example of categorical attribute: There is a finite number of types of items sold by the store (milk, butter, flower, sugar, …), and every type of item is essentially unrelated to the others. In our toy example, country and language are considered to be categorical attributes.

On the other hand, we can have numerical attributes. These take numerical values, as the name implies. For numerical attributes, rather than representing a set of distinct categories, the attribute represents a spectrum. Having values that are close represent roughly the same thing, while having values that are distant means that the two are quite different. The important distinction is that having, for example, 3 tweets is not *categorically different* to having 4

tweets. A user having 3 tweets is very similar (in terms of how many tweets they posted) to a user having 4 tweets, but these two are more similar to each other than to a third user having 1000 tweets. It is this continuous nature of numerical attribute which we concentrate on.

If we were to build the cuboid for the toy example on the "posts" attribute, it would give a very uninformative result (see Figure 2.3.1). The only vertices that are aggregated together are those that have exactly the same value for the attribute, completely disregarding the fact that values nearby (in the number line) are "nearly the same". Ideally we would like a way to reflect this in the aggregations of the cuboid.

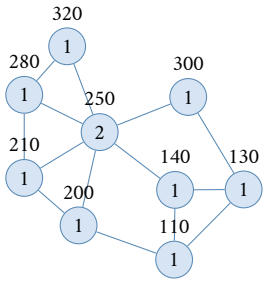


Figure 2.12: A very uninformative cuboid for (posts) for the toy network.

Note that there can be attributes considered to be *categorical* even though their values are numbers. One can imagine, in the context of characterizing insects, to consider the attribute "number of legs". This attribute takes numerical values, but if two values are close, it indicates in no way that the insects are similar. An insect with 6 legs is as different to an insect with 8 legs than to one with 30. Thus this attribute makes more sense as categorical than numerical.

2.3.2 Algorithm 1 is unsuited for numerical attributes

As exemplified in the previous section, using Algorithm 1 as-is does not give meaningful results for numerical attributes. Two vertices will only be aggregated together if they have exactly the same value. For example, a vertex with value 2.003 will not be grouped together with another vertex whose value is 2.004, since they are not strictly equal. These two will be considered "as different" as another vertex with value 300 for example.

The goal of this kind of pattern matching algorithm is to give an informative view of the dataset. For this, we would like the algorithm to have a way of grouping together values that are close into the same aggregate vertex, in a way that is the most informative for our dataset. This is the enterprise of this thesis.

The first idea that comes to mind is to apply a discretization algorithm to the numerical attributes before running Algorithm 1. Discretization algorithms take continuous numerical attributes and partitions them into a finite set of discretized "bins" — described by an interval. All the vertices will fall in exactly one of these bins. Every bin can then be thought of as a categorical attribute. Many different discretizing techniques could be used, for example Equal width or Equal frequency interval binning [9].

[9]: "Supervised and unsupervised discretization of continuous features", Dougherty, Kohavi, and Sahami, 1995

The idea of discretizing the data beforehand is good as a baseline, but it falls short on one important aspect. Our objective is finding the *most informative* grouping for the data, i.e. we want to form aggregated vertices which tell us the most about our dataset, and most importantly about the edges in our dataset. The graph structure is why we bother to use graph cuboids in the first place. If we discretize the numerical attributes with a binning algorithm before the search for patterns, it doesn't take the graph structure into account in the binning process.

What we want is a search that looks at different ways of aggregating vertices together, with the objective of building cuboids which are informative with respect to the edges present in the base network. This means that vertices that are grouped together should exhibit similar beheaviour in terms of the at-

tributes of the vertices they are (or aren't) connected to.

Once we have found an informative cuboid, we can use Florian's interestingness measure to find the interesting patterns in this cuboid. Alternatively, an informativeness measure can also be used to find which cuboid — i.e. which selection of attributes — leads to the "best" cuboid, in the categorical setting.

3

How informative is this cuboid? An answer based on stochastic blockmodels

As highlighted in the last chapter, the framework presented in [4] is not well suited to handle numerical attributes, because it has no way to aggregate them into meaningful condensed vertices. In order to search for the most informative aggregation for numerical attributes, we need a formal notion of what "informative" means in this context. This notion of informativeness is also missing in the original graph cube work. Cuboids are generated, but there is no formal notion of the quality of these aggregated views of the network.

[4]: "Graph cube mining", Demesmaeker, 2017

In this chapter, I will introduce a notion of quality of a cuboid, based on an adaptation of stochastic blockmodels to the context of graph cubes. This quality measure will be useful in the search of informative aggregations of the numerical attributes.

The big picture is this: Stochastic blockmodels give us a generative model of the connective structure of graphs, i.e. of what is connected to what. Given such a model, we can compute the likelihood that a particular connective structure was generated from the model. This measure tells us how much observed data (our multidimensional network) fits the model, or conversely, how much the model explains the data.

As I show in this chapter, there is a deep connection between stochastic blockmodels and graph cuboids. This connection allows us to consider graph cuboids as generative models for our base graph, and thus get a measure of how much the cuboid explains our dataset.

I start by presenting the notions of blockmodel and stochastic blockmodel in section 3.1. The relevant definitions are given, and then the parallels are drawn to translate between the language of stochastic blockmodels and the language of graph cuboids. Finally, a measure of informativeness of a cuboid is defined.

3.1 Stochastic blockmodels

Blockmodels, first introduced by White, Boorman, and Breiger [10], come originally from the study of social networks. They simplify large social graphs by partitioning the actors into blocks, allowing to examine only the interactions between blocks rather than all the individual interactions. Stochastic blockmodels, introduced by Holland, Laskey, and Leinhardt [11], are a stochastic generalization of blockmodels. They can be interpreted as gen-

[10]: "Social Structure from Multiple Networks. I. Blockmodels of Roles and Positions", White, Boorman, and Breiger, 1976

[11]: "Stochastic blockmodels", Holland, Laskey, and Leinhardt, 1983

erative models for graphs, and come with specialized estimation techniques and a likelihood measure. They are often used in the context of community detection [12].

[12]: "Community detection in general stochastic block models", Abbe and Sandon, 2015

I will start by giving the formal definition of stochastic blockmodels, as laid out in [11]. Then, I will explain how to learn the most likely stochastic blockmodel from a graph. Finally I will explain how the attributes in a multidimensional network can be used to define the blocks in a stochastic blockmodel.

3.1.1 Definition of a stochastic blockmodel

The following is taken from [11], with some minor modifications to clarify some points and make the notation more consistent with the rest of the thesis.

A stochastic blockmodel is a model for networks characterized by a block structure. By block structure, we mean that the nodes of the network are partitioned into subgroups called blocks, and that the distribution of the edges between vertices is dependent on the blocks to which the vertices belong. The model is a generalization of deterministic blockmodels, in a framework which allows for variability in the data.

More formally, a stochastic blockmodel is a special type of probability distribution over the space of adjacency matrices*. Let V be the set of $n = |V|$ vertices, and let $E \subseteq V \times V$ be a relation defined on the pairs of vertices.

* In [11], the definitions are given for stochastic blockmodels of multigraphs. Here, we restrict ourselves to single-relation graphs.

Definition 7. *The **adjacency matrix** for the digraph (V, E) is given by:*

$$\boldsymbol{x} = (x_{ij}) \text{ where } x_{ij} = \begin{cases} 1 & \text{if } (i,j) \in V; \\ 0 & \text{otherwise.} \end{cases}$$

Vectors and matrices will usually be denoted with bold type, like $\boldsymbol{x}$, while real values will be denoted with normal type. This is why we write $\boldsymbol{x} = (x_{ij})$ first in bold, then in regular type.

In what follows, we keep the convention of [11] and denote random quantities by uppercase letters, and particular realizations of the random quantities by lowercase letters. If $\boldsymbol{X}$ is a random adjacency matrix for n vertices and m edges, then the probability distribution of $\boldsymbol{X}$ is called a *stochastic graph*. We will denote the probability distribution of $\boldsymbol{X}$ by $\Pr(\boldsymbol{X} = \boldsymbol{x})$.

A stochastic blockmodel is a special case of a stochastic graph which satisfies the following requirements.

Definition 8. *Let $\Pr(\boldsymbol{X} = \boldsymbol{x})$ be the probability function for a stochastic graph, and let $\{B_1, \ldots, B_t\}$ be a partition of the vertices into mutually exclusive and exhaustive subsets† called "blocks".*

*We say that $\Pr(\boldsymbol{X} = \boldsymbol{x})$ is a **stochastic blockmodel** with respect to the partition $\{B_1, \ldots, B_t\}$ if and only if for any vertices $i \neq j$ and $i' \neq j'$, if i and i' are in the same block, and j and j' are in the same block, then the random variables X_{ij} and $X_{i'j'}$ are identically distributed.*

† $\bigcup_i B_i = V$, and $\forall i \neq j : B_i \cap B_j = \emptyset$

Definition 8 implies that vertices in the same block are stochastically equivalent in the following sense. Consider a block B_r and any vertex j in the network. The likelihood of any given pattern of edges with vertex j is the same for all vertices of block B_r.

Definition 9. *Let $\boldsymbol{X}$ be a random adjacency matrix. We say two vertices i and i' are **stochastically equivalent** if and only if the probability of any event about $\boldsymbol{X}$ is unchanged by interchanging vertices i and i'.*

We say that the pair of vertices (i, j) belongs to the *pair-block* $B_r \times B_s$ if and only if i is in the block B_r and j is in the block B_s. If $\Pr(\boldsymbol{X} = \boldsymbol{x})$ is the probability function for a stochastic blockmodel $\boldsymbol{X}$, then the *pair-distribution* for pair-block $B_r \times B_s$ is given by

$$p_{rs}(z) = \Pr(X_{ij} = z), \text{ for any } i \in B_r, j \in B_s, i \neq j, \tag{3.1}$$

where z is 0 or 1. Thus, $p_{rs}(1)$ is the probability that there is an edge between any vertex of block B_r and any vertex of block B_s. We call this quantity the *pair-block density* of $B_r \times B_s$ and denote it by π_{rs}.

$$\pi_{rs} = p_{rs}(1) = \Pr(X_{ij} = 1) \text{ for } i \in B_r, j \in B_s, i \neq j \tag{3.2}$$

The implied distribution of the matrix $\boldsymbol{X}$ is

$$\Pr(\boldsymbol{X} = \boldsymbol{x}) = \prod_{r,s} \prod_{i \in B_r} \prod_{\substack{j \in B_s \\ i \neq j}} \pi_{rs}^{x_{ij}} (1 - \pi_{rs})^{1 - x_{ij}} \tag{3.3}$$

Thus, a stochastic blockmodel over a partition $\{B_1, \ldots, B_t\}$ is completely determined by the $t \times t$ matrix of probabilities $\boldsymbol{\pi} = (\pi_{rs})$.

Let's illustrate with an example. Say we have a group of 3 persons, which we label $V = \{a, b, c\}$. They are partitionned into two "blocks": There are those who like chocolate ($B_1 = \{a, b\}$) and those who don't ($B_2 = \{e\}$). We consider the directed graph where an edge from x to y means that x likes y.

Let's imagine that we have a stochastic blockmodel over this partition. The $\boldsymbol{\pi}$ matrix is given in Figure 3.1. The probability of two chocolate-lovers being friends is given by $\pi_{11} = 100\%$. So all 4 chocolate lovers like each other. However, there is only a 10% probability that a cholocate lover likes a non-chocolate lover (π_{12}).

If we have a specific graph, we can calculate its probability under our stochastic blockmodel. Consider the graph in Figure 3.2. The probability of observing this graph with our blockmodel is the probability of its adjacency matrix in the implied distribution of the model:

$$\begin{aligned}
\Pr(\boldsymbol{X} = \boldsymbol{x}) &= \prod_{r=1}^{2} \prod_{s=1}^{2} \prod_{i \in B_r} \prod_{j \in B_s} \pi_{rs}^{x_{ij}} (1 - \pi_{rs})^{1 - x_{ij}} \\
&= \pi_{11}^2 (1 - \pi_{12})^2 \pi_{21} (1 - \pi_{21}) \\
&= 1^2 \cdot (1 - 0.1)^2 \cdot .75 \cdot (1 - .8) \\
&= 0.1215
\end{aligned}$$

	1	2
1	100	10
2	75	80

Figure 3.1: Example of $\boldsymbol{\pi}$ matrix for a stochastic blockmodel with only two blocks. The probabilities are given in % — i.e. there is a 100 % edge probability between two vertices of the first block, and a 10% edge probability between a vertex of the first block and a vertex of the second block.

$$\boldsymbol{x} = \left(\begin{array}{cc|c} 0 & 1 & 0 \\ 1 & 0 & 0 \\ \hline 1 & 0 & 0 \end{array}\right)$$

Figure 3.2: Example of directed graph with 3 vertices, along with its adjacency matrix, divided into blocks.

3.1.2 *Learning a stochastic blockmodel from a graph*

Let $G = (V, E)$ be a directed graph with adjacency matrix $\boldsymbol{x} = (x_{ij})$ whose vertices are partitioned into $\{B_1, \ldots, B_t\}$. We want to find the stochastic blockmodel for which $\boldsymbol{x}$ has maximal probability. In other words, we want to find the maximum likelihood estimates $\hat{\boldsymbol{\pi}}$ for the pair-block densities $\boldsymbol{\pi} = (\pi_{rs})$.

Let $b_r = |B_r|$ be the number of vertices in block B_r. Then the number of pairs in pair-block $B_r \times B_s$ is given by[‡]

[‡] In the case of undirected graphs, it is

$$b_{rs} = \begin{cases} \frac{1}{2} b_r (b_r - 1) & \text{if } r = s; \\ b_r b_s & \text{if } r \neq s. \end{cases}$$

$$b_{rs} = \begin{cases} b_r(b_r - 1) & \text{if } r = s; \\ b_r b_s & \text{if } r \neq s. \end{cases} \tag{3.4}$$

The -1 comes from the fact that we don't allow for self-loops in the graph.

Let $X_{++}(r,s)$[§] be the number of 1s in the $b_r \times b_s$ submatrix of $\boldsymbol{X}$ corresponding to pair-block $B_r \times B_s$, i.e. the number of edges from a vertex in B_r to a vertex in B_s

[§] The notation X_{++} indicates that we are doing a sum over all possible indices $i \neq j$ of X_{ij}

$$X_{++}(r,s) = \sum_{\substack{i \in B_r \\ i \neq j}} \sum_{j \in B_s} X_{ij}. \tag{3.5}$$

Then Equation 3.3 can be rewritten as

$$\Pr(\boldsymbol{X} = \boldsymbol{x}) = \prod_{r,s} \pi_{rs}^{x_{++}(r,s)} (1 - \pi_{rs})^{b_{rs} - x_{++}(r,s)}. \tag{3.6}$$

From Definition 8, it follows that the X_{ij} are independent Bernoulli random variables, where the Bernoulli parameter is π_{rs} if $(i,j) \in B_r \times B_s$. Therefore, the $X_{++}(r,s)$ are independent binomial (b_{rs}, π_{rs}) random variables, and the marginal distribution of $X_{++}(r,s)$ is given by

$$\Pr(X_{++}(r,s) = k) = \binom{b_{rs}}{k} (\pi_{rs})^k (1 - \pi_{rs})^{(b_{rs} - k)}. \tag{3.7}$$

Because $X_{++}(r,s)$ are independent binomial random variables, maximum likelihood estimation is particularly tractable for a stochastic blockmodel. Let $\boldsymbol{x}$ be the observed adjacency matrix arising from the stochastic blockmodel, where the blocks are given but $\boldsymbol{\pi} = (\pi_{rs})$ is unknown. Then the likelihood function for the data is

$$L(\boldsymbol{\pi}) = \prod_{rs} \pi_{rs}^{x_{++}(r,s)} (1 - \pi_{rs})^{b_{rs} - x_{++}(r,s)}. \tag{3.8}$$

Because the $X_{++}(r,s)$ are independent and there are no constraints on the π_{rs}, the maximum likelihood estimate for $\boldsymbol{\pi}$ is obtained by maximizing each of the marginal likelihood functions $L_{rs}(\pi_{rs}) = \pi_{rs}^{x_{++}(r,s)}(1 - \pi_{rs})^{b_{rs} - x_{++}(r,s))}$ with respect to π_{rs}. The maximum occurs at the sample proportions

$$\hat{\pi}_{rs} = X_{++}(r,s)/b_{rs}. \tag{3.9}$$

Thus, the maximum likelihood estimate $\hat{\boldsymbol{\pi}}$ of the matrix of block densities $\boldsymbol{\pi}$ is given by the matrix of observed block densities.

For example, let's calculate the maximum likelihood blockmodel for the small graph given in Figure 3.2. Remember that the block partition is $B_1 = \{a, b\}, B_2 = \{c\}$. The number of edges from B_1 to B_1 is 2. There are 2 possible edge, so $\pi_{11} = 2/2 = 1$. There are no edges from B_1 to B_2, so the pair-block density $\pi_{12} = 0$. There is one edge in pair-block $B_2 \times B_1$ (the edge from *c* to *a*), and $b_2 1 = 2 \cdot 1 = 2$, so $\pi_{21} = 1/2 = 0.5$. Finally, there are no edges from B_2 to B_2 so $\pi_{22} = 0$. The most likely matrix for π is thus given by

$$\hat{\boldsymbol{\pi}} = \begin{pmatrix} 1 & 0 \\ 0.5 & 0 \end{pmatrix}.$$

The probability of our graph under this model is $0.5 \cdot (1 - 0.5) = 0.25$.

3.1.3 Blockmodels in multidimensional networks

Let $\mathcal{N} = (V, E, A)$ be a multidimensional network, as defined in Definition 1. For any selection of attributes $S \subseteq A$, we can derive a partition of the set of vertices V. We first define the equivalence relation on vertices $\overset{S}{=} \subseteq V \times V$. Two vertices $u, v \in V$ are in this relation (written $u \overset{S}{=} v$) if they have the same values for every attribute in S:

$$u \overset{S}{=} v \iff \forall A_i \in S : A_i(u) = A_i(v).$$

Since $\overset{S}{=}$ is indeed an equivalence relation[¶], its quotient set $V / \overset{S}{=}$ forms a partition of the set of vertices. Remark that this is the same set as the set of aggregated vertices of the cuboid for aggregation S, as defined in Definition 2.

The same partition of vertices which defines the aggregated vertices in a cuboid can also be considered as the blocks of a blockmodel. It is the big takeaway from this section: every cuboid defines a blockmodel. I will now present the relation between the probabilistic model of patterns in [4] and that of stochastic blockmodels defined on cuboids. Then, using this relation, I will present a measure of informativeness for cuboids.

[¶] $\forall A_i \in S, \forall u, v, w \in V$:

- symmetric: $A_i(u) = A_i(v) \Rightarrow A_i(v) = A_i(u)$;
- reflexive: $A_i(u) = A_i(u)$;
- transitive: $A_i(u) = A_i(v) \wedge A_i(v) = A_i(w) \Rightarrow A_i(u) = A_i(w)$.

[4]: "Graph cube mining", Demesmaeker, 2017

3.2 Translating between the language of cuboids and the language of blockmodels

Since the theory of graph cubes and the theory of stochastic blockmodels were developed independently, they have adopted different notations for identical concepts. Here I present these similarities, and show where the two differ. I will methodically review the terminology of pattern mining in graph cubes and present their stochastic blockmodel equivalent. A handy overview is given in Table 3.1.

3.2.1 Comparing cuboids and stochastic blockmodels

Aggregated vertex In a cuboid for a selection of attributes S, an aggregated vertex $[v]$ is a vertex of the aggregate network — i.e. a vertex of the cuboid. It is the representative element of the aggregation of all the vertices from the base network which have the same values as v for the attributes in S, i.e. $[v] = \{v' \in V \mid S(v') = S(v)\}$. It can be referred to by the tuple of values $T_v = S(v)$. In the context of stochastic blockmodels, an aggregated vertex corresponds to a block. We write this block $B_{T_v} = \{u \in V \mid S(u) = T_v\}$.

Weight of an aggregated vertex In the aggregate network, the weight $w([v])$ of the aggregated vertex for T_v tells the number of vertices in the base network matching T_v in S, i.e. $w([v]) = |[v]| = |\{v' \in V \mid S(v') = T_v\}|$. It corresponds to the size of the block B_{T_v}, denoted by $b_{T_v} = |B_{T_v}|$.

Pattern A pattern P is defined by a selection of attributes S and two tuples of values (T_u, T_v) defined over these attributes. Every pattern is represented by an edge in the cuboid for S — in this case the edge $([u], [v])$. This corresponds to the pair-block $B_{T_u} \times B_{T_v}$ in the blockmodel.

Support of a pattern The support $\text{supp}(T_u, T_v)$ of the pattern $P = (T_u, T_v)$ is the number of edges in the base graph matching P, i.e. $|\{(u', v') \in E \mid S(u') = T_u \wedge S(v') = T_v\}|$. In the weighted graph of the corresponding cuboid, this value is found in the weight of the aggregated edge, $w([u], [v])$. This corresponds to $x_{++}(B_{T_u}, B_{T_v})$: the number of 1s in the $b_{T_u} \times b_{T_v}$ submatrix corresponding to pair-block $B_{T_u} \times B_{T_v}$.

Expected support Here, the partition model used in [4] differs from the stochastic blockmodel.

[4]: "Graph cube mining", Demesmaeker, 2017

In the former, the model assumes that we know the edges matching a more general pattern $P' = (T'_u, T'_v)$, defined over $S' \subset S$. If we sample randomly from these edges, the probability that the sampled edge also matches P is given by $\Pr(P \mid P') = p_{S,S'}(T_u)p_{S,S'}(T_v)$ where $p_{S,S'}(T_u)$ represents the proportion of vertices matching T'_u that also match T_u. In other words, we take all the vertices matching P' and consider their endpoints to be independently distributed into T_u and T_v according to the corresponding proportions. The expected support is then $E[\text{supp}_S(T_u, T_v)] = \text{supp}_{S'}(T_u, T_v)p_{S,S'}(T_u)p_{S,S'}(T_v)$.

In the latter, there is no prior knowledge about the number of edges. Instead, the model assumes that every couple of vertices in the pair-block $B_{T_u} \times B_{T_v}$ has the same probability $\pi_{T_uT_v}$ of being connected by an edge. Since there are $b_{T_uT_v}$ such couples, the expected support is then $E[x_{++}(T_u, T_v)] = b_{T_uT_v} * \pi_{T_uT_v}$.

Significance of a pattern Just as for the expected support, the significance of a pattern doesn't have an equal counterpart in the stochastic blockmodel. In the partition model for patterns, the significance of a pattern $P = (T_u, T_v)$ of S is defined as the probability of observing its support, given that the edges are drawn from the $|E'|$ edges in the more aggregated cuboid S' with probability $p = p_{S,S'}(T_u)p_{S,S'}(T_v)$:

$$\Pr(\text{supp}(P)) = \binom{|E'|}{\text{supp}(P)} p^{\text{supp}(P)}(1-p)^{|E'|-\text{supp}(P)}.$$

On the other hand, in the stochastic blockmodel, the probability of observing a certain support for a pattern is modelled as the selection of $\text{supp}(P)$ edges from $b_{T_uT_v}$ possible edges, with probability $\pi_{T_uT_v}$ (see Equation 3.7):

$$\Pr(\text{supp}(P)) = \binom{b_{T_uT_v}}{\text{supp}(P)} \pi_{T_uT_v}^{\text{supp}(P)}(1-\pi_{T_uT_v})^{b_{rs}-\text{supp}(P)}.$$

3.2.2 Recapitulation

3.3 How informative is a cuboid

Given that cuboids define blockmodels of the base graph, we can calculate, for any cuboid, its most likely stochastic blockmodel from the graph. Let $\boldsymbol{x}$ be the adjacency matrix of our multidimensional network $\mathcal{N} = (V, E, A)$, and let us consider a cuboid built on the selection of attributes $S \subset A$. This stochastic blockmodel is the probability distribution $\Pr(\boldsymbol{X} = \boldsymbol{x})$ over all possible adjacency matrices of graphs with the same partition $V/\overset{S}{=}$ for which the likelihood (Equation 3.8) is maximal.

cuboid			stochastic blockmodel	
$[v] = \{v' \in V \mid S(v') = T_v = S(v)\}$	aggregated vertex for tuple of values T_v	=	$B_{T_v} = \{u \in V \mid S(u) = T_v\}$	vertex block for tuple of values T_v
$w([v])$	weight of an aggregated vertex	=	b_{T_v}	size of a block
$P = (T_u, T_v)$	pattern (pair of aggregated vertices)	=	$B_{T_u} \times B_{T_v}$	pair-block
$\text{supp}_S(T_u, T_v) = w([u], [v])$	support of the T_u, T_v pattern (weight of the corresponding aggregated edge)	=	$x_{++}(B_{T_u}, B_{T_v})$	number of 1s in the $b_{T_u} \times b_{T_v}$ submatrix corresponding to pair-block $B_{T_u} \times B_{T_v}$
(no symbol)	number of possible edges between a vertex in $[u]$ and a vertex in $[v]$	=	$b_{T_u T_v}$	number of pairs of vertices in pair-block $B_{T_u} \times B_{T_v}$
$E[\text{supp}_S(T_u, T_v)]$	expected support of a pattern	$\neq$	$E[x_{++}(T_u, T_v)]$	expected value of $X_{++}(T_u, T_v)$
$\text{supp}_{S'}(T_u, T_v) p_{S,S'}(T_u) p_{S,S'}(T_v)$		$\neq$	$b_{T_u T_v} * \pi_{T_u T_v}$	
$\Pr(\text{supp}(P))$	significance of a pattern	$\neq$	$\Pr(X_{++}(T_u, T_v) = k)$	marginal distribution of $X_{++}(T_u, T_v)$
$\binom{\lvert E' \rvert}{\text{supp}(P)} p^{\text{supp}(P)} (1-p)^{\lvert E' \rvert - \text{supp}(P)}$		$\neq$	$\binom{b_{T_u T_v}}{k} (\pi_{T_u T_v})^k (1 - \pi_{T_u T_v})^{(b_{T_u T_v} - k)}$	

Table 3.1: A Rosetta stone for cuboids and stochastic blockmodels.

The expression of this probability distribution is of the form

$$\Pr(\boldsymbol{X} = \boldsymbol{x}) = \prod_{r,s} \pi_{rs}^{x_{++}(r,s)} (1 - \pi_{rs})^{b_{rs} - x_{++}(r,s)},$$

where the pair-block densities π_{rs} are given by Equation 3.9 : $\pi_{rs} = x_{++}(r,s)/b_{rs}$.

The value of this expression gives a measure of how well the cuboid captures the graph structure of the graph. Indeed, a high probability means that the base network is very likely given the information in the cuboid. In other words, the cuboid contains a lot of information about the structure of the network. Such a cuboid is said to be "informative" of the base graph. Conversely, a cuboid for which the stochastic blockmodel probability is low doesn't capture much of the structure of the graph, and is considered "uninformative".

We call this value the **informativeness** of the cuboid of S for the network $\mathcal{N}$, and denote it by $\text{info}_{\mathcal{N}}(S)$.

Definition 10. *Let $\mathcal{N} = (V, E, A)$ be a multidimensional network, $S \subset A$ be a selection of attributes, and $G' = (V', E', W_{V'}, W_{E'})$ be the aggregate network with regards to S. Let $\boldsymbol{x}$ be the adjacency matrix of $\mathcal{N}$. The **informativeness** of the cuboid S for the network $\mathcal{N}$ is given by*

$$\text{info}(S) = \prod_{([u],[v]) \in V'} p^{w([u],[v])} (1-p)^{b_{T_u T_v} - w([u],[v])},$$

where $p = w([u], [v])/b_{T_u T_v}$.

Another way to consider this value is to take the logarithm of its inverse $\log_2(\frac{1}{\text{info}_{\mathcal{N}}(S)})$. Since informativeness is a probability measure, the logarithm of its inverse is its *surprisal* (or self-information [13]). This is the number of bits needed to transmit the whole adjacency matrix, given the knowledge of

[13]: *Elementary information theory*, Jones, 1979

the cuboid S and a minimum-length encoding of the message. A low number of bits means that the cuboid "compresses" the graph well, i.e. it contains useful information about the structure of the adjacency matrix.

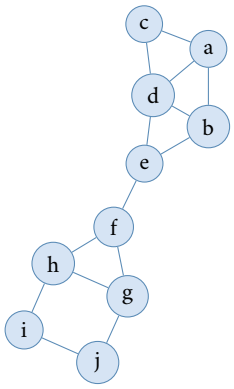
Definition 11. *The **surprisal** of a cuboid S with respect to a multidimensional network $\mathcal{N}$ is the amount of bits needed to encode the adjacency matrix of the network, using a perfect encoding based on the knowledge of the cuboid.*

It is given by $\text{surprisal}_{\mathcal{N}}(S) = -\log_2(\text{info}_{\mathcal{N}}(S))$.

In this way, we can say that the cuboid with the highest info score — i.e. the lowest surprisal — is the "best" cuboid, under the *mimimum description length* principle (MDL). It is the cuboid whose associated stochastic blockmodel leads to the best compression of the data.

This is related to the concept of *Occam's razor*: All other things being equal, the best explanation is the one which is the simplest.

For example, let's construct the stochastic blockmodel for the (country) cuboid in our toy example, and calculate its informativeness. The base graph is shown in Figure 3.3 and the cuboid in Figure 3.4.



Vertex	Country	Language	Posts
a	BE	en	280
b	NL	en	210
c	FR	fr	320
d	BE	en	250
e	NL	nl	200
f	BE	nl	110
g	BE	fr	130
h	NL	en	140
i	FR	fr	250
j	FR	fr	300

Figure 3.3: Toy example of a multidimensional network with its graph and multidimensional vertex attribute table.

Remember that every block B_r in the blockmodel corresponds to an aggregated vertex v. The weight of this vertex $w(v)$ is the number of vertices in the aggregation, which is also the size of the block b_i. Similarly, every pair-block $B_r \times B_s$ in the blockmodel corresponds to a pair of condensed vertices (v, w) in the cuboid. The weight of an edge (v, w) is the number of edges between vertices corresponding to v and vertices corresponding to w, which is also $x_{++}(r, s)$ — the number of 1s in the $b_r \times b_s$ submatrix of $\boldsymbol{x}$ corresponding to pair-block $B_r \times B_s$.

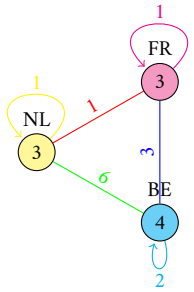


Figure 3.4: (country) cuboid for the toy example.

Let's consider the edge between a and b from the base graph. Vertex a is Belgian, and b is dutch. In our cuboid, the edge BE–NL has weight of 6. Since there are $4 \cdot 3 = 12$ couples of the type BE-NL, the probability of observing an edge between a and b is

$$\Pr(x_{\text{BE,NL}} = 1) = \pi_{\text{BE,NL}} = \frac{x_{++}(\text{BE, NL})}{b_{\text{BE, NL}}} = \frac{w(\text{BE,NL})}{b_{\text{BE,NL}}} = \frac{6}{12}$$

Using the same logic, we can calculate the probability of observing the whole base graph under the blockmodel described by the cuboid by multiplying the probabilities of observing each edge (or absence of edge) in the base graph:

$$\Pr(\boldsymbol{X} = \boldsymbol{x}) = \prod_{r,s} \prod_{i \in B_r} \prod_{\substack{j \in B_s \\ i \neq j}} \pi_{rs}^{x_{ij}} (1 - \pi_{rs})^{1 - x_{ij}}.$$

We have already remarked that, since edges in the same pair-block have the same probability (by the definition of stochastic blockmodels), we can group them together in this expression to get

$$\Pr(\boldsymbol{X} = \boldsymbol{x}) = \prod_{r,s} \pi_{rs}^{x_{++}(r,s)} (1 - \pi_{rs})^{b_{rs} - x_{++}(r,s)},$$

which is the same as the product over all aggregated vertices of the cuboid

$$\prod_{u,v \in V'} \left(\frac{w(u,v)}{b_{uv}} \right)^{w(u,v)} \left(1 - \frac{w(u,v)}{b_{uv}} \right)^{b_{uv} - w(u,v)}.$$

The relevant values for our toy example are shown in Figure 3.5, along with the toy example network, color-coded as in the (country) cuboid in Figure 3.4. The probability of observing the base graph according to the stochastic blockmodel is calculated as follows:

$$\begin{aligned}
\text{info}(\text{(country)}) &= \Pr(\boldsymbol{X} = \boldsymbol{x}) \\
&= \prod_{r,s} \pi_{rs}^{x_{++}(r,s)} (1 - \pi_{rs})^{b_{rs} - x_{++}(r,s)} \\
&= (1/3)^1 (1 - 1/3)^{3-1} && \text{(FR,FR)} \\
&\quad (2/6)^2 (1 - 2/6)^{6-2} && \text{(BE,BE)} \\
&\quad (1/3)^1 (1 - 1/3)^{3-1} && \text{(NL,NL)} \\
&\quad (3/12)^3 (1 - 3/12)^{12-3} && \text{(FR,BE)} \\
&\quad (1/9)^1 (1 - 1/9)^{9-1} && \text{(FR,NL)} \\
&\quad (6/12)^6 (1 - 6/12)^{12-6} && \text{(BE,NL)} \\
&\approx 5.975 \cdot 10^{-12}.
\end{aligned}$$

# of pairs b_{rs}	# of edges $x_{++}(r,s)$	pair-block r,s	proba. π_{rs}
3	1	FR FR	1/3
6	2	BE BE	2/6
3	1	NL NL	1/3
12	3	FR BE	3/12
9	1	FR NL	1/9
12	6	BE NL	6/12

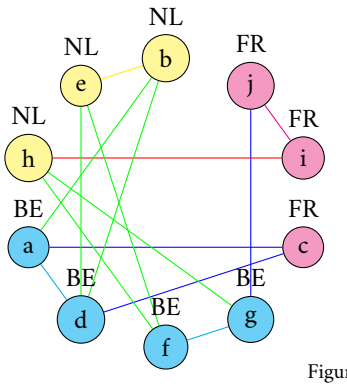


Figure 3.5: Calculation worksheet for the stochastic blockmodel probability of the (country) aggregation of the toy example.

If we were to do the same calculations for the (language) cuboid (shown in Figure 3.6, we would get $\text{info}(\text{(language)}) \approx 1.4293 \cdot 10^{-11}$. This means that the (language) cuboid is about 2.4 times more informative than the (country) cuboid.

Or, if we look in terms of surprisal, the (country) cuboid has a surprisal of $\approx$ 37 bits and the (language) cuboid has a surprisal of $\approx$ 36 bits. If we were to compress the adjacency matrix of the graph using a perfect compression scheme that uses the prior knowledge of one of these cuboids, it would take one more bit to transmit the graph using the (country) cuboid than using the (language) cuboid.

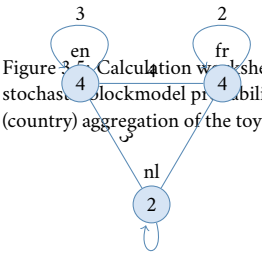


Figure 3.6: (language) cuboid of the toy network.

3.4 Using the informativeness measure to handle numerical attributes

So far, in this chapter, we have presented a way to construct a stochastic blockmodel from a cuboid. With this stochastic blockmodel, we have defined a measure of the informativeness of the cuboid, $\mathsf{info}(S)$. This measure indicates how much information the cuboid contains about the graph structure of the network.

Our motivation for devising such a measure was to guide the search for informative aggregations over numerical attributes. Indeed, since numerical attributes have no canonical way of being aggregated into meaningful groups, we needed a way to measure if one grouping of vertices based on their numerical values is better than another.

The $\mathsf{info}(S)$ measure gives us exactly that. By comparing the $\mathsf{info}(S)$ of the cuboids built on different aggregations, we can determine which one has the highest score, i.e. which one captures the most information about the graph structure of the base network. We will now apply this and develop a search procedure.

4
Informative cuboids with numerical binning

Consider the following setting: we are given a multidimensional network with both categorical and numerical attributes. We wish to see an aggregated view of our network, where the vertices are grouped together according to a selection of attributes. Some of these attributes are categorical, and others are numerical. Furthermore, we want for this aggregated view to give us insight about the edges in our network.

For example, let's consider our toy example and the selection of attributes (country, posts). If most of the Belgians with more than 200 posts and most of the Dutchmen with more than 200 posts are connected together, we would like that to be reflected in our aggregated view.

One approach to achieve this is to discretize the numerical attribute into bins — for example put all the vertices with less than 200 posts in one bin, and those with 200 posts or more in another — in such a way that the cuboid built on these bins along the other categorical attributes is the most informative. The measure of informativeness is given by $\mathrm{info}(S)$.

This is the approach that will be explored in this chapter. In section 4.1 I will define what we mean by binning and present two baseline binning approaches. Then, in section 4.2, I will present a greedy heuristic for local search in the space of possible binnings. Finally, in section 4.3, I will present an experimental setup in which to validate and analyze the performance of this approach for the search of informative cuboids.

4.1 Baseline: Discretizing beforehand

The idea of this approach is to divide the range of values of the numerical attribute into "bins" — that is, separate the range of possible values into a series of consecutive, non-overlapping intervals. The number of bins can vary, and they are not required to be of equal size.

Once this is done, we can transform the numerical attribute into a categorical one. We do this by replacing each numerical value by the bin that contains it. This way, we have only categorical attributes, and we can build the cuboid in the traditional way.

The number of bins $n_{\text{bins}}$is considered to be a parameter of the algorithm, and will not be changed during the search. The search-space will be the different sizes that the n_{bins}bins can take

The baseline for this kind of search is not to search at all. In order to verify that our search finds an interesting discretization, we will compare it to the

two most common binning approaches: eq-width binning and eq-frequency binning.

4.1.1 Equi-width binning

In eq-width binning, every bin has the same width, i.e. every interval has the same size. For example, if we bin an attribute with values from 0 to 100 into 4 bins of equal width, the intervals will be $[0, 25[$, $[25, 50[$, $[50, 75[$ and $[75, 100]$*. Every bin has a equal width of $100/4 = 25$.

This is the simplest binning method, as it only depends on the minimum and maximum value of the samples being binned. However it has the limitation that some of the bins might be empty, if no samples fall in a given interval.

* Here, the choice is made that every interval includes its starting point, and excludes its ending point, with the exception of the last interval. Other equivalent choices can be made, but the impact of this convention choice is negligible.

4.1.2 Equi-frequency binning

In eq-frequency binning, every bin contains the same number of samples. The bins are still defined by intervals, but they no longer have to have the same width. For example, if $n_{\text{bins}} = 2$, and there are 50 vertices with values between 0 and 1, and 50 vertices with values between 1 and 5, then the eq-frequency bins will be $[0, 1[$, $[1, 5]$. Both bins will contain 50 vertices, even if the second one is four times wider than the first one.

In eq-frequency binning, it is not always possible to have exactly the same number of samples in each bin, because multiple samples can have the same value. Consider the following case: the samples are $0, 0, 0, 4$. If we bin these into 2 bins with the intervals $[0, 2[$ and $[2, 4]$, the first bin has 3 samples and the second bin has 1 sample. They do not all have the same number of samples. It is obvious that there is no way to have two bins of size 2, as this would mean separating the three 0 from each other. This happens rarely enough that it can be ignored for our purposes.

4.1.3 Binning in practice

I will now show how these binning techniques will be used to generate cuboids with numerical and categorical attributes. Let S be the selection of attributes on which we wish to build the cuboid. We denote by C the categorical attributes in S, and by N the numerical attributes in S, with $C \cup N = S$.

We first group the vertices according to their categorical attributes, i.e. we build the partition $\left(V / \overset{C}{=}\right) = \{V_1, \ldots, V_t\}$. Then, for each set of vertices belonging to the same category V_i, we discretize each numerical attribute $N_j \in N$ into n_{bins}bins. The numerical value for N_j of each vertex in V_i is then replaced by the bin it falls into, thus transforming the numerical attribute into a categorical one.

Let's illustrate with an example. Consider again our toy network. We will build the cuboid on the (country, posts) aggregation, using eq-width discretization for the numerical attribute "posts". We select $n_{\text{bins}} = 2$. The relevant information is shown in Figure 4.1.

First we separate the vertices according to the categorical attributes, here "country". We have $\left(V / \overset{C}{=}\right) = \{V_{\text{BE}}, V_{\text{NL}}, V_{\text{FR}}\}$, with $V_{\text{BE}} = \{\text{a,d,f,g}\}$, $V_{\text{FR}} = \{\text{c,i,j}\}$, and $V_{\text{NL}} = \{\text{b,e,h}\}$.

Then, for every V_i, we find the 2 bins.

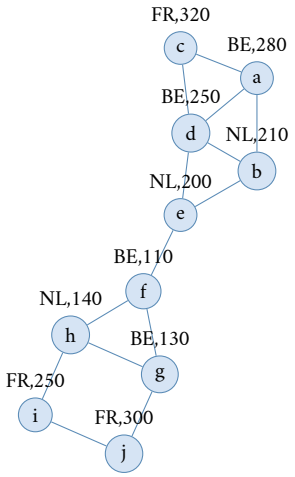


Figure 4.1: Toy network, with the value for attributes "country" and "posts" indicated for every vertex.

- The number of posts of vertices in V_{BE} range from 110 to 280, so the two bins are $[110, 195[$ and $[195, 280]$. Vertices {f,g} fall in the first bin, and {a,d} in the second bin;
- For V_{FR}, the values range from 250 to 320, so the bins are $[250, 285[$ and $[285, 320]$. Vertex i falls in the first bin, and vertices {c,j} fall in the second bin.
- Finally, for V_{NL}, the values range from 140 to 210, so the bins are $[140, 175[$ and $[175, 210]$. Vertex h falls in the first bin, and vertices {b,e} fall in the second bin.

The last step is to build the cuboid, using the discretized bins instead of the numerical attribute. This is shown in Figure 4.2.

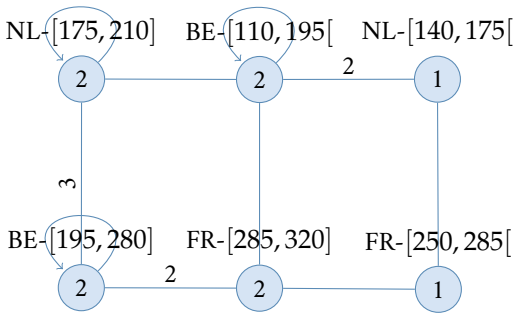


Figure 4.2: (country, posts) cuboid for the toy network, with the numerical attribute "posts" discretized into 2 equal-width bins.

4.1.4 *Discussion*

Equi-width and eq-frequency binning give us a good baseline, and may give informative results (especially if n_{bins} is large). However, if they do give informative results, it is only "by luck". The bins are blind guesses, created without any consideration for the graph structure of the network, which is what will ultimately determine whether the discretization makes an informative cuboid or not.

In the next section, I will present a greedy algorithm that improves on these blind guesses by repeatedly changing the intervals in the way which improves the most the informativeness of the cuboid.

4.2 *Greedy algorithm*

The space of all divisions of n samples into n_{bins} bins is very large. Indeed, in the worst case scenario (all samples having different numerical values), the number of possible binnings is equal to $p_{n_{\text{bins}}}(n)$, the number of compositions of n with exactly n_{bins} parts.

Lets define exactly what we mean. We learn from [13] that a composition of n (also called a *partition* of n) is an expression of n as an *ordered sum* of positive integers. For instance, there are eight compositions of 4; namely

$$1+1+1+1; \quad 2+1+1; \quad 1+2+1; \quad 1+1+2;$$
$$3+1; \quad 1+3; \quad 2+2; \quad 4.$$

If exactly k summands appear in a composition σ, we say that σ has k parts. There is a bijection between all k-compositions of n and and the number of

[13]: *Elementary information theory*, Jones, 1979

binnings of n elements into k bins, when all the elements have different values. The idea is simple: given a k-composition of n (for example, for $k = 3, n = 4$, $\sigma = 1 + 2 + 1 = 4$), you can form 3 bins for 4 elements by having the first bin contain the first element, the second bin contain the next 2 elements, and the last bin contain the last 1 element. In the other direction of the bijection, given a binning of n elements into k bins, the size of the bins in order give a k-composition of n — their sum must be equal to n since every of the n elements is in a bin.

The number of k-compositions of n is given by

$$p_k(n) = \binom{n-1}{k-1} = \frac{(n-1)!}{(k-1)!(n-k)!},$$

so the number of different binnings for n samples and n_{bins}bins is $\frac{(n-1)!}{(n_{\text{bins}}-1)!((n-n_{\text{bins}})!}$. This means that an exhaustive search would consider $O(n^{n_{\text{bins}}-1})$ different binnings. This quickly becomes prohibitive as n_{bins}becomes bigger.

Instead of considering every binning in the search space, we will perform a local search starting from an existing binning. We will visualize the binnings as a list of $n_{\text{bins}} - 1$ pivots. Every pivot is a sample which defines where one bin ends and the next one starts. All the samples with value lower than the pivot go in the previous bin, and the ones with value greater or equal go in the following bin. A visualization is shown in Figure 4.3.

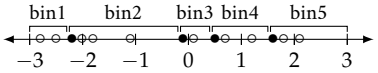


Figure 4.3: A visualization of binning with pivots. The samples are shown as dots on the number line. The full dots are the pivots for our bins, and the empty dots are regular samples. Every pivot defines the boundary between two bins.

Given a certain binning — i.e. a list of split points — the neighbouring binnings are all those obtained by moving one of the split points to the next (or previous) sample in the list. The greedy heuristic is the following: We start from a given binning. We consider all the neighbouring binnings, and move to the one which improves the $\text{info}(S)$ score the most. We repeat this until we reach a local optimum.

4.2.1 Pseudocode

The pseudocode for this algorithm is given in Algorithm 2. Remark that, when calculating the delta of scores between two cuboids $\text{info}(S') - \text{info}(S)$, since $\text{info}(\cdot)$ is a product over all couples of aggregated vertices

$$\prod_{([u],[v])\in V'} p^{w([u],[v])}(1-p)^{b_{T_u T_v} - w([u],[v])},$$

parts of the product will be equal in S' and in S. When moving a pivot by only one place, we effectively only move one vertex from one bin into another bin. The resulting cuboid differ only in the aggregated vertices corresponding to those two bins. When neither $[u]$ nor $[v]$ are one of the changed bins, then the expression $p^{w([u],[v])}(1-p)^{b_{T_u T_v} - w([u],[v])}$ will be identical for S and S'.

If instead of using $\text{info}(\cdot)$ as our measure, we use $\log(\text{info}(\cdot))$, then the product becomes a sum, and all the identical expressions in the delta calculation cancel out. This is why we use the logarithm of the informativeness in our algorithm: It allows us to simplify the calculation of the delta. If we denote by V'' the set of aggregated vertices of the S' cuboid, and the two aggregated vertices being changed by t, f and t', f' (in the S and S' cuboid respectively),

the expression of $\Delta(S, S') = \log \text{info}(S') - \log \text{info}(S)$ becomes

$$\sum_{\substack{([u],[v])\in V'' \\ [u] \text{ or } [v]\in\{t',f'\}}} \log\left(p^{w([u],[v])}(1-p)^{b_{T_u T_v}-w([u],[v])}\right) - \sum_{\substack{([u],[v])\in V' \\ [u] \text{ or } [v]\in\{t,f\}}} \log\left(p^{w([u],[v])}(1-p)^{b_{T_u T_v}-w([u],[v])}\right).$$

4.2.2 Limitations and possible improvements

This is a greedy search which will stop at the first local optimum found. There is no guarantee that the binning returned is optimal in any other capacity. We can try to mitigate this by repeating the algorithm with random initial binnings, and take the best result.

Another approach could be to do a sort of simulated annealing, i.e. to have a temperature parameter, which decreases over the iterations of the algorithm, and which dictates how much "random movements" through the search space we should add.

4.3 Validation on synthetic data

We will now analyze how the binning approach behaves in a more realistic setting. We will generate graphs based on a realistic model. These graphs will then be aggregated into discretized cuboids with the algorithms presented previously in this chapter. We will then investigate how appropriate and informative the cuboids are with respect to the generating model.

This kind of validation was also missing from the work in [4]. The experiments there were done on a real-life dataset (the MovieLens dataset, from [14]). Using a dataset from a real application setting allows to see if the search for pattern reveals some interesting patterns about the domain of the application. However, since we do not have perfect knowledge of the dataset and the application domain, there is no way of verifying if the algorithm missed some patterns. There is also no verification of overfitting.

[4]: "Graph cube mining", Demesmaeker, 2017

[14]: "The MovieLens Datasets", Harper and Konstan, 2015

We will address these shortcomings specifically in the case of our binning approach on numerical attributes. Using a probabilistic model, we will generate graphs on which to run the algorithm. Since the generating model will be known exactly, we will be able to assess exactly the performance of our algorithm. Furthermore, we can generate multiple graphs from the same model, build the cuboid on one graph and verify it on the others, in order to see if the cuboid was overfitting the particular structure of the graph it was built on, or if it actually captured the underlying model structure.

4.3.1 The generative model for multidimensional networks with numerical attributes

I will now describe the generative model for a multidimensional network $\mathcal{N} = (V, E, A)$, where A contains some categorical attributes C and some numerical attributes N, with $A = C \cup N$.

We consider that we have a number of different populations $\mathcal{P} = \{\mathcal{P}_1, \dots \mathcal{P}_t\}$. Every vertex in our network will come from one of these populations.

Every population $\mathcal{P}_i$ has a fixed tuple of values for the categorical attribute, $T_{\mathcal{P}_i} = C(p), \forall p \in \mathcal{P}_i$. We call this tuple the *category of* $\mathcal{P}_i$.

Algorithm 2: Greedy search for representative binnings

```
1: function GREEDY(A, B, P)
2:     Input: A the adjacency matrix between the aggregated vertices
3:     Input: B the size of the blocks for each c ∈ V / ≡^C
4:     Input: P the pivots for each c ∈ V / ≡^C
5:     Output: The pivots for a cuboid a local optimum
6:     do
7:         bestAction ← NONE ▷ this will store the best action found so far.
8:         bestΔ ← 0        ▷ This will store the best score delta found so far.
9:         for c ∈ V / ≡^C, pivot ∈ P(c) do
10:            for direction ∈ {LEFT, RIGHT} do
11:                if CANMOVE(pivot, direction) then
12:                    if DELTA(pivot, direction) > bestΔ then
13:                        bestΔ ← DELTA(pivot, direction)
14:                        bestAction ← MOVE pivot TO THE direction
        EXECUTE(bestAction)
15:    while bestAction ≠ NONE
16: function DELTA(pivot, direction)
17:    build the cuboid C based on the current pivots
18:    build the cuboid C′ with the current pivots but with the specified
    pivot moved in the specified direction
19:    return log info(C′) − log info(C)
20: function CANMOVE(pivot, direction)
21:    if direction = RIGHT then
22:        return false if it is already at the rightmost position or if it is next
    to the following pivot, and true otherwise
23:    if direction = LEFT then
24:        return false if it is already at the leftmost position, or if it is next
    to the previous pivot and true otherwise
25: function EXECUTE(action)        ▷ Updates the A, B, P data structures to
    effectively move the given pivot in the given direction.
26:    movedVertex ← the vertex that is being moved from one bin to an-
    other
27:    fromBin ← the bin that the vertex is currently in
28:    toBin ← the bin that the vertex will be put in
29:    decrease B(fromBin) by 1
30:    increase B(toBin) by 1
31:    for every edge (u, v) ∈ V do
32:        if u = movedVertex then
33:            otherBin ← the bin of v
34:            decrease A(fromBin, otherBin) by 1
35:            increase A(toBin, otherBin) by 1
36:        if v = movedVertex then
37:            otherBin ← the bin of u
38:            decrease A(otherBin, fromBin) by 1
39:            increase A(otherBin, toBin) by 1
40:    change the value of the pivot in P
```

Every population has a fixed tuple of numerical distributions for the numerical attributes, $\mathcal{D}_{\mathcal{P}_i}$. The value for attribute N_j of every vertex from population $\mathcal{P}_i$ will be sampled from the corresponding distribution $\mathcal{D}_{\mathcal{P}_i j}$.

Every population $\mathcal{P}_i$ also has a specific probability π_{ij} of having an edge to another population $\mathcal{P}_j$. This means that if u is from population $\mathcal{P}_i$ and v is from population $\mathcal{P}_j$, the probability of observing an edge from u to v is given by π_{ij}.

Thus, a generative model with t different populations is completely defined by

- t tuples of values $T_{\mathcal{P}_i}$ defined over the categorical attributes C;
- t tuples of numerical distributions $\mathcal{D}_{\mathcal{P}_j}$ defined over the numerical attributes N;
- a $t \times t$ matrix of inter-population edge densities $\boldsymbol{\pi} = (\pi_{ij})$.

The procedure to generate a multidimensional network from a model is as follows:

1. select a random population $\mathcal{P}_i$;
2. create a new vertex v in the graph, with its categorical attributes given by $T_{\mathcal{P}_i}$, and its numerical attributes sampled from the distributions $\mathcal{D}_{\mathcal{P}_j}$;
3. for every other vertex w in the network, recover its population $\mathcal{P}_j$, and generate an edge (v, w) with probability π_{ij}. Then, generate an edge (w, v) with probability π_{ji};
4. repeat until the desired number of vertices have been generated.

Note that since every vertex belongs to a population and the inter-population dentities are fixed, this model also qualifies as a stochastic blockmodel. Indeed, vertices from the same population are stochastically equivalent. The blocks of the stochastic blockmodel are defined by the different populations, and the inter-block densities are given by the inter-population densities $\boldsymbol{\pi}$.

4.3.2 Experimental scenarios

For the experiment, I have designed 4 different generating models to compare the performance of the algorithm in different scenarios.

In order for the results to be easier to analyze, the models contain only one categorical attribute and one numerical attribute. The models are all comprised of 6 populations. The numerical attribute will take three different values ("a", "b" and "c"). Since the focus is the handeling of numerical attributes, our population should not be entirely separable through the categorical attribute. To this end, $\mathcal{P}_1$ and $\mathcal{P}_2$ have category ("a"), $\mathcal{P}_3$ and $\mathcal{P}_4$ have category ("b"), and $\mathcal{P}_4$ and $\mathcal{P}_5$ have category ("c"). The numerical distributions will all be Gaussians, but they will be different in the different scenarios. The inter-population densities are different in the four scenarios.

First scenario For the first scenario, the mixtures for the two populations in every category are well-separated — i.e. they do not overlap much. They are shown in Figure 4.4. The inter-population densities range from 0% to 10%.

They were chosen so that two populations with the same category have very different connection patterns. This means that, for example, the probabilities of drawing an edge from population 1 to another population π_{1*} will be very different to π_{2*}. The $\boldsymbol{\pi}$ matrix is shown in Figure 4.6.

Second scenario For this scenario, there are only two distributions. The two distributions overlap a lot. For every pair of population with the same category, one has its numerical attribute drawn from the first distribution and the other one from the second distribution — i.e. $\mathcal{D}_{\mathcal{P}_1} = \mathcal{D}_{\mathcal{P}_3} = \mathcal{D}_{\mathcal{P}_5} = \mathcal{D}_1$ and $\mathcal{D}_{\mathcal{P}_2} = \mathcal{D}_{\mathcal{P}_4} = \mathcal{D}_{\mathcal{P}_6} = \mathcal{D}_2$. The distributions are shown in Figure 4.5. The $\boldsymbol{\pi}$ is the same as in scenario 1, shown in Figure 4.6.

Third scenario The third scenario uses the same well-separated distributions as the first scenario (Figure 4.4) but the inter-population edge dentities are much more similar between two populations of the same category. Differences there are of the order of one or two percents. The matrix for $\boldsymbol{\pi}$ is shown in Figure 4.7.

Fourth scenario The fourth scenario uses the overlapping distributions of scenario 2 (Figure 4.5) with the inter-population densities of scenario 3 (Figure 4.7).

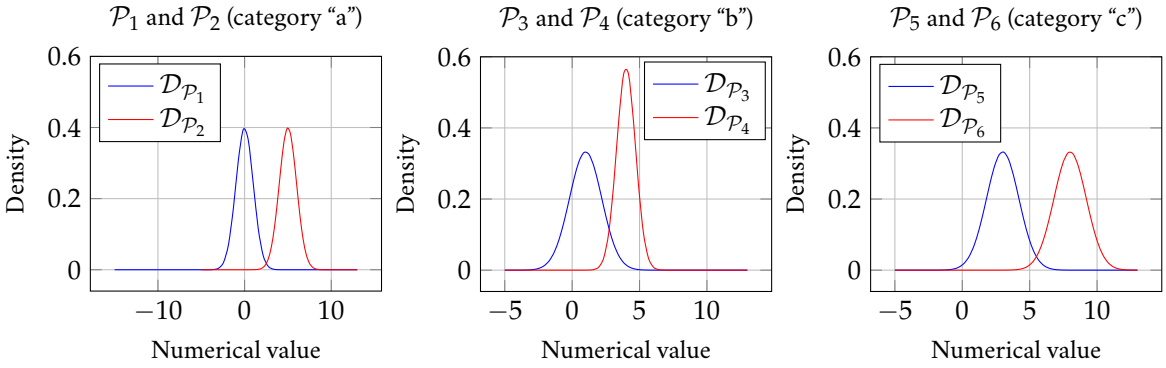


Figure 4.4: Well-separated gaussian distributions for the 6 populations of scenarios 1 and 3.

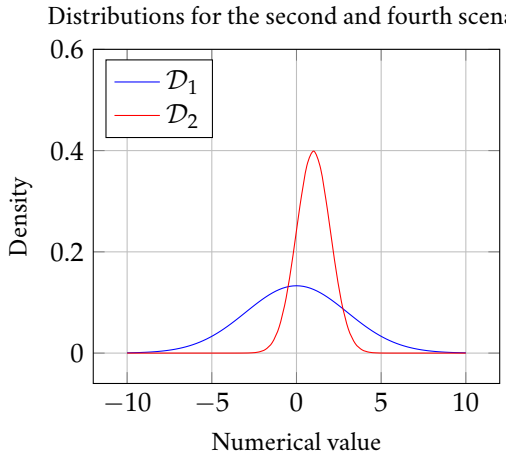


Figure 4.5: Overlapping distributions for scenarios 2 and 4.

	1	2	3	4	5	6
1	6	9	0	8	7	0
2	1	1	7	0	0	8
3	0	0	8	0	0	0
4	0	0	0	3	0	0
5	8	6	0	0	1	5
6	0	0	8	5	5	0

Figure 4.6: π matrix for scenarios 1 and 2. The value at row i and column j is the probability (in percent) of drawing an edge between a vertex of population $\mathcal{P}_i$ and one of $\mathcal{P}_j$. The values are in percent.

	1	2	3	4	5	6
1	4	5	6	4	0	0
2	5	4	7	4	0	0
3	0	0	1	2	4	4
4	0	0	2	1	3	3
5	7	5	0	0	3	2
6	6	7	0	0	2	2

Figure 4.7: π matrix for scenarios 3 and 4. The value at row i and column j is the probability (in percent) of drawing an edge between a vertex of population $\mathcal{P}_i$ and one of $\mathcal{P}_j$. The values are in percent.

4.3.3 *Experimental setup*

First experiment From these models, I generated a number of graphs. Every vertex in the generated graph has a hidden attribute, which indicates from which population this vertex was drawn. Since we know the population to which each vertex belongs, we can easily build the cuboid aggregating on the populations. This is the ideal cuboid for this graph, since it groups together the vertices perfectly, according to their behaviour. We can then get the $\mathrm{info}()$ score of the ideal cuboid with respect to the stochastic blockmodel of the generating scenario, which we call the ideal informativeness.

When running our binning algorithms, the population to which each vertex belongs will be hidden. The algorithm will discretize the numerical attribute, and build a cuboid on the categorical and discretized attributes of the vertices. Then, we will compare the informativeness of the generated cuboid to the ideal informativeness. The closer we are to this ideal value, the better.

We will compare the performance of the equi-width and equi-freq binning, as-is and with the greedy optimization. We will also observe the effect of varying n_{bins} from 2 to 5. The results for generated graphs with 1000 vertices are shown in Table 4.1. In order to increase readability, we will use the surprisal score rather than the info score, so lower values mean better cuboids[†].

[†] Remember that $\mathrm{surprisal}(\cdot) = -\log(\mathrm{info}(\cdot))$.

We can see that, in general, the score is better for scenarios 3 and 4 than for scenario 1 and 2. This is explained by the fact that, in those two scenarios, the inter-population densities are quite similar for populations having the same category. This makes it so that misclassifying a vertex induces less error. Conversely, in scenarios 1 and 2, the populations sharing the same category have very different inter-population densities, which induces a higher penalty for aggregating together vertices from different populations.

We also observe, as expected, that the greedy search always improves the score of the generated cuboid, though not always by much. Since we are binning the vertices based on their numerical value, we cannot expect to completely separate them by population due the overlap in the generating numerical distributions. Unless we get very lucky, the numerical values of two populations sharing the same categorical attribute will not be separable into

	scenario 1		scenario 2		scenario 3		scenario 4	
ideal:	100,200		99,914		108,392		107,258	
n_{bins}	eq-width	greedy	eq-width	greedy	eq-width	greedy	eq-width	greedy
2	104,766	103,800	115,810	115,735	108,524	108,437	107,640	107,640
3	105,951	103,119	116,480	115,487	108,571	108,560	107,612	107,605
4	103,591	103,226	115,184	115,082	108,493	108,458	107,567	107,569
5	103,936	103,075	114,662	114,053	108,413	108,403	107,546	107,537
n_{bins}	eq-freq	greedy	eq-freq	greedy	eq-freq	greedy	eq-freq	greedy
2	104,130	103,774	116,556	116,571	108,458	108,437	107,654	107,653
3	107,923	103,197	115,568	115,500	108,566	108,567	107,613	107,610
4	103,537	102,666	115,170	115,222	108,416	108,405	107,563	107,561
5	105,403	103,054	114,618	114,647	108,449	108,456	107,512	107,499

Table 4.1: surprisal score of the cuboids generated by the baseline binning algorithms and the greedy algorithm starting from this baseline. All values are in bits.

two intervals. What's more, the greedy approach has no guarantee of finding the optimal binning. Still, it is good to see that this approach can sometimes improve the score by as much as 1Kbit.

For the baseline approaches in scenarios 1 and 3, we can also observe that the score is better for $n_{\text{bins}} = 2$ than for $n_{\text{bins}} = 3$. The explanation for this is that every category is a mix of two different populations, whose values are well-separated. Splitting the vertices into 2 groups according to their value is a good way of separating the two populations. However, splitting into three groups makes a middle group which will inevitably be a mix of the two populations, which leads to worse scores. This is not the case for scenarios 2 and 4, whose distributions overlap. For those two scenarios, adding more bins increases the granularity of the generated cuboids and leads to better scores.

Second experiment: avoiding overfitting One big shortcoming of the previous experiment is that we calculate the score of a cuboids on the graphs that were used to generate that cuboid. This mean that the cuboid could be modelling parts of the graph that are not representative of actual differences between the populations, but just specific features of the specific graph it was built on.

This is a classical case of overfitting. Since we have the generating models for the datasets, we can observe the impact of overfitting with the following experiment: We generate a number of new graphs from the same generating model, and calculate the scores of the cuboid built previously on these new graphs. If the cuboid was overfitting the graph, we will have a worse score from this experiment than in the previous experiment.

We ran into a small issue for this experiment. It can happen that, in our cuboid, there is no edge between two aggregated vertices, for example, beween $[u]$ with T_1 =("a", [0, 10[) and $[v]$ with T_2 =("b", [-5, 0[)). The probability of observing an edge between these two aggregated vertices is then $\pi_{T_1T_2} = \frac{w([u],[v])}{w([u])w([v])} = \frac{0}{w([u])w([v])} = 0$. However, in a second graph, we have an edge between two vertices u', v' with attributes respectively ("a", 1) and ("b", -1). u' will be aggregated into $[u]$ and v' will be aggregated into $[v]$, since their value fall in the corresponding intervals. However, the probability of this edge is 0!

The probability of observing the whole graph under this cuboid thus becomes 0, since there is a 0 in the product.

In order to circumvent this issue, we added additive smoothing (also called Laplace smoothing, [15]). Since the $X_{++}(r,s)$ are independent binomial random variables (see Equation 3.7), the smoothed versions of the data gives the estimator

[15]: *Introduction to Information Retrieval*, Manning, Raghavan, and Schütze, 2008

$$\hat{\pi}_{rs} = \frac{X_{++}(r,s) + \alpha}{b_{rs} + 2\alpha} = \frac{w([u],[v]) + \alpha}{b_{T_1T_2} + 2\alpha}, \tag{4.1}$$

where α is the smoothing parameter, here chosen to be 1. This smoothing has the effect of pushing the probabilities closer to 0.5. Most importantly, no probability will be 0: Even if $w([u],[v]) = 0$, the probability $1/(b_{T_1T_2} + 2)$ will be non-negative.

In this experiment, we generated 20 graphs of 2000 vertices for every scenario. The results are shown in Table 4.2. For every cuboid generated in the previous experiment, we show the mean and standard deviation of the score on the 20 graphs. The ideal mean and deviation are calculated from the ideal cuboid, as for the previous experiment. Entries have $\mu = +\infty, \sigma =$ NaN when some probabilities were too small to fit in a Double value. For every setting, the best score is shown in bold.

		scenario 1		scenario 2		scenario 3		scenario 4	
ideal:	μ	394,875		393,279		425,841		425,650	
	σ	4627		4486		3482		3679	
n_{bins}		eq-width	greedy	eq-width	greedy	eq-width	greedy	eq-width	greedy
2	μ	415,971	410,819	**457,093**	456,558	**426,427**	**426,295**	**427,381**	**427,383**
	σ	4387	4627	4974	4813	3476	3496	3665	3663
3	μ	419,514	409,072	460,748	455,729	426,755	426,743	427,848	427,481
	σ	4742	4827	4306	4681	3513	3512	3665	3660
4	μ	**410,217**	**408,331**	$+\infty$	**453,899**	426,538	426,535	$+\infty$	427,403
	σ	4305	4600	NaN	4867	3476	3482	NaN	3652
5	μ	$+\infty$	408,515	$+\infty$	454,006	426,591	426,662	$+\infty$	428,025
	σ	NaN	4747	NaN	5093	3487	3482	NaN	3602
n_{bins}		eq-freq	greedy	eq-freq	greedy	eq-freq	greedy	eq-freq	greedy
2	μ	412,344	412,024	460,251	460,294	**426,295**	**426,295**	427,521	427,517
	σ	4737	4860	4784	4779	3496	3496	3670	3670
3	μ	427,091	408,700	456,551	456,551	426,813	426,816	427,420	**427,424**
	σ	4264	4770	4795	4764	3505	3506	3663	3661
4	μ	**410,102**	**407,471**	454,791	454,725	426,428	426,437	**427,416**	427,460
	σ	4638	4803	4904	4935	3498	3498	3651	3650
5	μ	$+\infty$	408,087	**454,086**	**453,952**	426,776	426,809	427,496	427,546
	σ	NaN	4805	4960	4910	3512	3513	3645	3643

Table 4.2: surprisal score of the cuboids generated by the baseline binning algorithms and the greedy algorithm starting from this baseline, cross-validated on 20 graphs of 2000 vertices generated from the same model. All values are in bits. The best value of n_{bins} for every case is printed in bold.

The standard deviations for this example are all on par with the ideal standard deviations. The scores are all proportionally worse than in the previous experiment (about +1% worse), which was expected and shows that there was some overfitting.

We can see even more clearly the effect of overfitting in the equi-width case. The best score is nearly always for $n_{\text{bins}} = 2$. For high values of n_{bins}, the baseline equi-width discretization is especially bad, as we see many $+\infty$ scores. However, quite surprisingly, the greedy optimization will nearly always improve the score. This means that it really does capture some of the underlying populations, and not just patterns specific to the graph it is running on.

The scores for the equi-freq approach are on par with the scores in the equi-width approach, and often a bit better. Here however, the greedy approach is no longer improving on the score. We have much less $+\infty$ scores, mostly because this approach never generates bins with 0 samples, which lead to very small probabilities.

4.4 Discussion

In this chapter, we have shown how to discretize the numerical attributes of a multidimensional network into bins, and how to calculate the informativeness of the resulting cuboids. Then, we presented a greedy algorithm to improve on such a binning.

Finally, we presented an experimental setup to see the performance of this algorithm in a semi-realistic setting, using a generative model for multidimensional graphs. We have shown that this approach gives interesting results, while not optimal. We have also shown that the resulting cuboids tend to overfit the data.

In practice, graph cube techniques are used in settings where we have one big dataset, but we cannot generate more data. In that case, cannot generate more graphs in order to validate that a cuboid does not overfit the data. We could still partition the dataset into smaller subgraphs, and use one to train the cuboid and the others to validate. However in that case, if the graph is connected, we will loose some information about edges going from one subgraph to the other.

All in all, unless our populations have distributions which are completely separated, this approach can never hope to generate perfect results. Still, binning is gives results which are easy to understand — aggregated vertices are described by categorical attributes and intervals — and easy to produce. In cases where we expect the populations to have very little overlap in their numerical attributes, this approach would make sense.

5

Informative cuboids with mixture models and EM

Gaussian distributions are a natural way of thinking about random variables with real values. They arise from many natural processes, they maximize the entropy and have important mathematical properties (like the central limit theorem) which make them a very common default modelling choice [16]. Notably, we have chosen to use Gaussian distributions to model the values that numerical attributes take in the generative model presented in subsection 4.3.1.

[16]: *Statistical inference*, Casella and Berger, 2002

In this chapter, we make the assumption that our generative model fits the multidimensionnal network being analyzed — i.e. that the graph is composed of different populations, each having a different connection pattern to other populations, and each having their numerical attributes drawn from a Gaussian distribution. With this assumption, the problem of finding a representative cuboid becomes the problem of detecting the underlying populations. Once the populations are known, we could aggregate the vertices based on the population to which they belong, and get an informative cuboid.

If we had no graph structure and no categorical attributes — i.e. if we had only numerical values taken from different Gaussian distributions — then the problem of detecting the populations boils down to detecting the different distributions for each population. Since samples are drawn from a number of Gaussian distributions, the distribution of all the samples is a *mixture of Gaussians*. Detecting the different distributions for each population is then finding the *maximum likelihood estimate* (MLE) for the parameters of the mixture of Gaussians model.

This problem is usually solved with an expectation-maximization algorithm (EM, [17]). In this chapter, I will formalize these ideas and present an EM approach to find informative cuboids. In section 5.1, I present a model for Gaussian mixtures with latent variables. From this base, I describe a latent variable model for multidimensional networks. I explain why this first model is too complex for EM, and proceed to simplify it. In section 5.2, I will present the EM algorithm for Gaussian mixtures, and adapt it for our latent variables graph model. Finally, in section 5.4, I discuss the advantages and shortcomings of this approach.

[17]: "Maximum Likelihood from Incomplete Data via the EM Algorithm", Dempster, Laird, and Rubin, 1977

5.1 A statistical model for multidimensionnal networks using latent variables

The notation and definitions in this chapter follow those from chapter 9 of [18], with some modification for clarity and consistency.

[18]: *Pattern recognition and machine learning*, Bishop, 2006

5.1.1 Latent variables and Gaussian mixture models

We will start by defining the terms *latent variable* and *gaussian mixture model*.

A latent variable (also called hidden variable) is a variable of a model which cannot be directly observed. They are opposed to observable variables, which can be directly measured.

A Gaussian mixture model is a simple linear superposition of Gaussian components. It can be written in the form

$$\Pr(X = x) = p(x) = \sum_{k=1}^{K} \tau_k N(x|\mu_k, \sigma_k^2). \tag{5.1}$$

Each of the K Gaussian densities in the Gaussian mixture distribution

$$N(x|\mu_k, \sigma_k^2) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left\{-\frac{1}{2\sigma^2}(x-\mu)^2\right\}$$

is called a *component* of the mixture and has its own mean μ_k and variance σ_k^2. It is scaled by a *mixing coefficient* τ_k. The parameters $\{\tau_k\}$ must satisfy

$$0 \leq \tau_k \leq 1 \text{ as well as } \sum_{k=1}^{K} \tau_k = 1 \tag{5.2}$$

in order to be valid probabilities. An example of a Gaussian mixture with 3 components is shown in Figure 5.1.

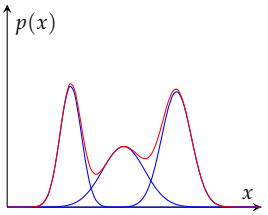


Figure 5.1: Example of gaussian mixture distribution with 3 components (each scaled by a coefficient, shown in blue) adn their sum (shown in red).

We can formulate a Gaussian mixture in terms of discrete latent variables, as in [18]. We introduce a K-dimensional binary random variable $\boldsymbol{z}$ having a 1-of-K representation in which a particular element z_k is equal to 1 and all other are equal to 0. The values of z_k therefore satisfy $z_k \in \{0, 1\}$ and $\sum_k z_k = 1$, and we see that there are K possible states for the vector $\boldsymbol{z}$ according to which element is nonzero. We shall define the joint distribution $p(x, \boldsymbol{z})$ in terms of a marginal distribution $p(\boldsymbol{z})$ and a conditional distribution $p(x|\boldsymbol{z})$,

$$p(x, \boldsymbol{z}) = p(\boldsymbol{z})p(x|\boldsymbol{z}).$$

The marginal distribution over $\boldsymbol{z}$ is specified in terms of the mixing coefficients τ_k, such that $p(z_k = 1) = \tau_k$. Because $\boldsymbol{z}$ uses a 1-of-K representation, we can also write this distribution in the form

$$p(\boldsymbol{z}) = \prod_{k=1}^{K} \tau_k^{z_k}.$$

Similarly, the conditional distributions of x given a particular value for $\boldsymbol{z}$ is a Gaussian $p(x|z_k = 1) = N(x|\mu_k, \sigma_k^2)$ which can also be written in the form

$$p(x|\boldsymbol{z}) = \prod_{k=1}^{K} N(x|\mu_k, \sigma_k^2)^{z_k}$$

The joint distribution is given by $p(\boldsymbol{z})p(x|\boldsymbol{z})$, and the marginal distribution of x is given by summing the joint distribution over all possible states of $\boldsymbol{z}$, giving

$$p(x) = \sum_{\boldsymbol{z}} p(\boldsymbol{z})p(x|\boldsymbol{z}) = \sum_{k=1}^{K} \tau_k N(x|\mu_k, \sigma_k^2),$$

which corresponds to the Gaussian mixture we were expecting (Equation 5.1). If we have several observations $x_1, \ldots, x_n$, then it follows that for every observed data point x_i there is a corresponding latent variable $\boldsymbol{z}_i$.

This formulation of the mixture model in terms of latent variables allows us to work with the joint distribution $p(x, \boldsymbol{z})$ instead of the marginal distribution $p(x)$. This allows us to define the following quantity $\gamma(z_k) = p(z_k = 1|x)$, whose value can be found using Bayes' theorem

$$\begin{aligned}\gamma(z_k) = p(z_k = 1|x) &= \frac{p(z_k = 1)p(x|z_k = 1)}{\sum_{j=1}^{K} p(z_j = 1)p(x|z_j = 1)} \\ &= \frac{\tau_k N(x|\mu_k, \sigma_k^2)}{\sum_{j=1}^{K} \tau_k N(x|\mu_j, \sigma_j^2)}. \end{aligned} \tag{5.3}$$

This quantity can be viewed as the *responsibility* that component k takes for "explaining" observation x*.

* Again, this comes from [18].

5.1.2 *A latent variable model for multidimensional networks with numerical attributes*

Let's consider the simple case of a mixture model $\mathcal{N} = (V, E, A)$ with only one numerical attribute and one categorical attribute. Thus, for every vertex i in our network, we can observe its numerical attribute x_i, its categorical attribute c_i, its outgoing edges $\boldsymbol{g}_{i*}$ and its incoming edges $\boldsymbol{g}_{*i}$†. These are the *observable* variables of our model.

† In this chapter, we write the adjacency matrix with the letter $G = (g_{ij})$ — for *g*raph — instead of x to avoid confusion with the numerical values and keep the notation consistent with the mixture of Gaussians model.

Our model will be a mixture of K populations. Every population $\mathcal{P}_k$ has a categorical attribute χ_k, a mixture coefficient τ_k and a Gaussian distribution for the numerical attribute $N(x|\mu_k, \sigma_k^2)$. The model also has a $K \times K$ inter-population edge density matrix, $\boldsymbol{\pi} = (\pi_{ij})$, which gives the probability of observing an edge between $\mathcal{P}_i$ and $\mathcal{P}_j$, $\forall i, j \in V$. We denote by V_k the subset of vertices having categorical attribute χ_k.

The number of populations K and the categories of each population χ_k are known in advance, but the rest of the model has to be learned from the observed data. We will model the distribution using latent variables $\boldsymbol{z}_i$, with the same one-of-K representation as for the Gaussian mixture model. For a vertex i, the latent variable $z_{ik} = 1$ if i comes from population $\mathcal{P}_i$.

Remark that the mixture coefficients τ_k only have to add up to 1 for the populations sharing the same category. We say that for every category χ_k

$$\sum_{\substack{k'=1 \\ \chi_k = \chi_{k'}}}^{K} \tau_{k'} = 1. \tag{5.4}$$

This also means that the probability $p(z_{ik} = 1)$ is no longer simply equal to τ_k. When vertex i has the same categorical attribute as population $\mathcal{P}_k$, then it

is τ_k, but otherwise it is 0. We define τ_{ik} such that

$$\tau_{ik} = p(z_{ik} = 1) = \begin{cases} \tau_k & \text{if } c_i = \chi_k; \\ 0 & \text{otherwise.} \end{cases}$$

If we know $\boldsymbol{z}_i$ (the population from which i was generated), the probability of observing x_i is given by the Gaussian distribution of this population

$$p(x_i|z_{ik} = 1) = N(x_i|\mu_k, \sigma_k^2). \tag{5.5}$$

If we know $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$, the probability of observing an edge g_{ij} is given by the edge densities

$$p(g_{ij}|z_{ik} = 1, z_{jk'} = 1) = \pi_{kk'}^{g_{ij}}(1 - \pi_{kk'})^{(1-g_{ij})}. \tag{5.6}$$

The joint probability of observing a vertex i if we know all the latent variables $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n$ is given by the probability of observing its numerical attribute and all its edges, $p(x_i, \boldsymbol{g}_{*i}, \boldsymbol{g}_{i*}|\boldsymbol{z}_*) =$

$$\begin{aligned} & p(x_i|z_{ik} = 1) \prod_{j \neq i} p(g_{ij}|z_{ik} = 1, z_{jk'} = 1) p(g_{ji}|z_{ik} = 1, z_{jk'} = 1) \\ = \quad & N(x_i|\mu_k, \sigma_k^2) \prod_{j \neq i} \pi_{kk'}^{g_{ij}}(1 - \pi_{kk'})^{1-g_{ij}} \pi_{k'k}^{g_{ji}}(1 - \pi_{k'k})^{1-g_{ji}}. \end{aligned}$$

Here we can already see a fundamental difference between the mixture of Gaussians model for numerical values and the mixture of populations model for multidimensional networks. In the former, the conditional probability for x_i is only dependent on the latent variable $\boldsymbol{z}_i$, and we can write the expression of $p(x_i|\boldsymbol{z}_i)$ simply (Equation 5.5). In the latter, the conditional probability for a vertex i depends on all the latent variables $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n$. Because of this, we no longer get a simple expression of the responsibility of each component $\gamma(z_{nk})$ — Equation 5.3 for the Gaussian mixture model — which is indispensable for the EM algorithm.

Because of this, the model just described is too complex for inference using the EM algorithm. We will simplify the model in order to get a workable expression, and develop an EM algorithm on this simplified model.

5.1.3 Simplifying the model

We chose the following simplification: Instead of considering every vertex comes from a population, and that all the edges are dependent on the (already fixed) populations of its end points, we consider every edge as independent from the vertices. The populations we select for the end points of the edge no longer have to match the population of the vertices.

With this simplification, we can separate the probability of each edge, as in Equation 5.6. The probability of observing a particular edge is given by

$$\begin{aligned} p(g_{ij}) &= \sum_{\boldsymbol{z}_i} \sum_{\boldsymbol{z}_j} p(\boldsymbol{z}_i) p(\boldsymbol{z}_j) p(g_{ij}|\boldsymbol{z}_i, \boldsymbol{z}_j) \\ &= \sum_{k=1}^{K} \sum_{k'=1}^{K} \tau_{ik} \tau_{jk'} \pi_{kk'}^{g_{ij}}(1 - \pi_{kk'})^{1-g_{ij}}, \end{aligned}$$

and the probability of observing a particular numerical value is given by

$$p(x_i) = p(\boldsymbol{z}_i) p(x_i|\boldsymbol{z}_i) = \sum_{k=1}^{K} \tau_{ik} N(x_i|\mu_k, \sigma_k^2).$$

Since edges and vertices are now independent, their joint probability is the product of these expressions.

$$p(x_i, g_{ij}) = p(x_i)p(g_{ij})$$

The likelihood of a given network with adjacency matrix $\boldsymbol{g}$, numerical values $\boldsymbol{x}$ and categorical values $\boldsymbol{c}$ under this simplified model with parameters $\boldsymbol{\theta} = (\boldsymbol{\tau}, \boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\pi})$ is

$$\begin{aligned} L(\boldsymbol{X}, \boldsymbol{G}|\boldsymbol{\theta}) &= \prod_{i \in V} p(x_i) \prod_{i,j \in V} p(g_{ij}) \\ &= \prod_{i \in V} \sum_{k=1}^{K} \tau_{ik} N(x_i|\mu_k, \sigma_k^2) \prod_{i \neq j \in V} \sum_{k=1}^{K} \sum_{k'=1}^{K} \tau_{ik}\tau_{jk'} \pi_{kk'}^{g_{ij}} (1 - \pi_{kk'})^{(1-g_{ij})} \end{aligned} \tag{5.7}$$

5.2 Expectation Maximization algorithm for Gaussian mixtures

We will now present the Expectation Maximization (EM) algorithm for Gaussian mixtures. The EM algorithm is a powerful method for finding maxmimum likelihood solutions for models with latent variables. We will first present the EM algorithm in the context of finding the maximum likelihood parameters for a Gaussian mixture model from some observations. This will give a good introduction to the relevant mechanisms of EM algorithms.

Then, in the next section, we will adapt this algorithm to find the maximum likelihood solutions for our simplified multidimensional network model. Since this model contains Gaussian mixtures, the calculations from the first EM algorithm will be useful.

This subsection also follows from [18], with some minor modifications. Suppose we have a dataset of observations $\{x_1, \ldots, x_n\}$. The latent variable model for Gaussian mixtures — given in subsection 5.1.1 — has the following likelihood function:

$$L(\boldsymbol{X}|\boldsymbol{\tau}, \boldsymbol{\mu}, \boldsymbol{\sigma}) = \prod_{i=1}^{n} \sum_{k=1}^{K} \tau_k N(x_n|\mu_k, \sigma_k^2).$$

Fitting the model to the dataset means finding the values for $\boldsymbol{\tau}, \boldsymbol{\mu}, \boldsymbol{\sigma}$ which maximize this likelihood function. The expression for this maxmium is easier to find if we consider the log-likehihood

$$\ell(\boldsymbol{X}|\boldsymbol{\tau}, \boldsymbol{\mu}, \boldsymbol{\sigma}) = \sum_{i=1}^{n} \log \left(\sum_{k=1}^{K} \tau_k N(x_n|\mu_k, \sigma_k^2) \right).$$

Setting the derivatives of ℓ with respect to the means μ_k to zero, we obtain[‡]

$$0 = -\sum_{i=1}^{n} \frac{\tau_k N(x_i|\mu_k, \sigma_k^2)}{\sum_{k'} \tau_{k'} N(x_i|\mu_{k'}, \sigma_{k'}^2)} \frac{x_i - \mu_k}{\sigma_k^2}.$$

[‡] Here we use the following identity:

$$\frac{\partial}{\partial \mu} N(x|\mu, \sigma^2) = N(x|\mu, \sigma^2) \frac{x - \mu}{\sigma^2}$$

Remark that the responsibilities $\gamma(z_{ik})$ (Equation 5.3) appear naturally in this expression. Multiplying by σ_k^2 on both sides and rearranging, we obtain

$$\mu_k = \frac{1}{N_k} \sum_{i=1}^{n} \gamma(z_{ik}) x_i, \tag{5.8}$$

where we have defined $N_k = \sum_{i=1}^{n} \gamma(z_{ik})$.

If we set the derivative of ℓ with respects to the deviation σ_k to zero, and follow a similar line of reasoning, we get

$$\sigma_k^2 = \frac{1}{N_k} \sum_{i=1}^{n} \gamma(z_{ik})(x_i - \mu_k)^2 \tag{5.9}$$

These have the same form as the corresponding results for a single Gaussian distribution fitted to the dataset, but with each data point weighted by the corresponding posterior probabilities and with the denominator given by the effective number of points associated with the corresponding component.

Finally, we maximize ℓ with respect to the mixing coefficients τ_k. Here we must take into account the constraint of Equation 5.2, which requires the mixing coefficient to sum up to one. This can be achieved using a Lagrange multiplier and maximizing the following quantity

$$\ell(\boldsymbol{X}|\boldsymbol{\tau}, \boldsymbol{\mu}, \boldsymbol{\sigma}) + \lambda \left(\sum_{k=1}^{K} \tau_k - 1 \right),$$

which gives

$$0 = \sum_{i=1}^{n} \frac{N(x_i|\mu_k, \sigma_k^2)}{\sum_{k'} \tau_{k'} N(x_i|\mu_{k'}, \sigma_{k'}^2)} + \lambda$$

where again we see the appearance of the responsibilities. If we now multiply both sides by τ_k and sum over k by making use of Equation 5.2, we find $\lambda = -n$. Using this to eliminate λ and rearranging we obtain

$$\tau_k = \frac{N_k}{n} \tag{5.10}$$

Equations 5.8, 5.9 and 5.10 are not closed forms which we can directly evaluate, since the responsibilities $\gamma(z_{ik})$ depend on those parameters in a complex way —see Equation 5.3. However, they suggest a simple iterative scheme for finding a solution to the maximum likelihood problem, which happens to be the EM algorithm for Gaussian mixtures. We will choose some initial values for $\boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\tau}$. Then, we alternate between these two steps:

1. **Expectation step (E)**: we use the current values for the parameters to evaluate the responsibilities, given by Equation 5.3.
2. **Maximization step (M)**: we re-estimate the means, deviations, and mixing coefficients using Equation 5.8, Equation 5.9 and Equation 5.10.

Every iteration is guaranteed to increase the likelihood function ([18]). This is the EM algorithm for Gaussian mixtures, which is recapitulated in Algorithm 3.

5.3 EM for our latent variable model

We will now adapt the EM algorithm to our simplified model for multidimensional networks. The likelihood function of our model was given in Equation 5.7. We take the log of this expression to get

$$\ell(\boldsymbol{\theta}) = \sum_{i \in V} \log \left(\sum_{k=1}^{K} \tau_{ik} N(x_i|\mu_k, \sigma_k^2) \right) + \sum_{i,j \in V} \log \left(\sum_{k=1}^{K} \sum_{k'=1}^{K} \tau_{ik} \tau_{jk'} \pi_{kk'}^{g_{ij}} (1 - \pi_{kk'})^{(1-g_{ij})} \right)$$

Algorithm 3: Pseudocode for the EM algorithm for Gaussian mixtures

Choose some initial values for $\boldsymbol{\mu}$, $\boldsymbol{\sigma}$ and $\boldsymbol{\tau}$.
repeat
  E step:
  for $i \leftarrow 1, \ldots, n, k \leftarrow 1, \ldots, K$ **do**
    $\gamma(z_{ik}) \leftarrow \frac{\tau_k N(x_i|\mu_k,\sigma_k^2)}{\sum_{k'=1}^{K} \tau_{k'} N(x_i|\mu_{k'},\sigma_{k'}^2)}$, as in Equation 5.3.
  M step:
  for $k \leftarrow 1, \ldots, K$ **do**
    $N_k \leftarrow \sum_{i=1}^{n} \gamma(z_{ik})$
    $\mu_k \leftarrow \frac{1}{N_k} \sum_{i=1}^{n} \gamma(z_{ik}) x_i$, as in Equation 5.8.
    $\sigma_k^2 \leftarrow \frac{1}{N_k} \sum_{i=1}^{n} \gamma(z_{ik})(x_i - \mu_k)^2$, as in Equation 5.9.
    $\tau_k \leftarrow \frac{N_k}{n}$, as in Equation 5.10.
until the parameters $\boldsymbol{\mu}$, $\boldsymbol{\sigma}$ and $\boldsymbol{\tau}$ or ℓ converges

Let Q_i be some distribution over the z's, with the sum over all k sharing the same categorical attribute $\sum_{\substack{k'=1 \\ \chi_k=\chi_{k'}}} Q_i(k) = 1$ and $Q_i(k) \geq 0$. This is analogous to the $\gamma(z_{ik})$ in the previous subsection. Let Q_{ij} be defined similarly over couples of z's, with $\sum_{\substack{k_1k_2 \\ \chi_{k_1}=\chi_{k_2}=\chi_k}} Q_{ij}(k_1,k_2) = 1, Q_{ij}(k_1,k_2) \geq 0$.

In order to be more concise, we will write

- $\sum_k$ to mean $\sum_{k=1}^{K}$;
- $\sum_{k_1k_2}$ to mean $\sum_{k1=1}^{K} \sum_{k_2=1}^{K}$.
- $\sum_i$ to mean $\sum_{i=1}^{n}$;
- $\sum_{ij}$ to mean $\sum_{i=1}^{n} \sum_{\substack{j=1 \\ i \neq j}}^{n}$;

The function to maximize in the M step is:

$$\max_{\theta} \sum_i \sum_k Q_i(k) \log \frac{\tau_{ik} N(x_i|\mu_k,\sigma_k^2)}{Q_i(k)} \tag{5.11}$$

$$+ \sum_{ij} \sum_{k_1k_2} Q_{ij}(k_1,k_2) \log \frac{\tau_{ik_1}\tau_{jk_2}\pi_{k_1k_2}^{g_{ij}}(1-\pi_{k_1k_2})^{(1-g_{ij})}}{Q_{ij}(k_1,k_2)} \tag{5.12}$$

We will now derive the closed-form formulas for these maximizations.

5.3.1 E step: estimating the posterior distributions

As in EM for Gaussian mixtures, the distribution $Q_i(k)$ is given by $p(z_{ik} = 1|x_i;\boldsymbol{\theta})$:

$$\begin{aligned} Q_i(k) &= p(z_{ik} = 1|x_i;\boldsymbol{\theta}) \\ &= \frac{p(z_{ik}=1)p(x_i|z_{ik}=1)}{\sum_{k'} p(z_{ik'}=1)p(x_i|z_{ik'}=1)} && \text{using Bayes' theorem} \\ &= \frac{\tau_{ik} N(x_i|\mu_k,\sigma_k^2)}{\sum_{k'} \tau_{ik'} N(x_i|\mu_{k'},\sigma_{k'}^2)} \end{aligned} \tag{5.13}$$

The Q_{ij} are found similarly:

$$\begin{aligned} Q_{ij}(k_1,k_2) &= p(z_{ik_1} = 1, z_{jk_2} = 1|g_{ij};\boldsymbol{\theta}) && \text{using Bayes' theorem} \\ &= \frac{p(z_{ik_1}=1)p(z_{jk_2}=1)p(g_{ij}|z_{ik_1}=1,z_{jk_2}=1)}{\sum_{k_1',k_2'} p(z_{ik_1'}=1)p(z_{jk_2'}=1)p(g_{ij}|z_{ik_1'}=1,z_{ik_2'}=1)} \\ &= \frac{\tau_{ik_1}\tau_{jk_2}\pi_{k_1k_2}^{g_{ij}}(1-\pi_{k_1k_2})^{(1-g_{ij})}}{\sum_{k_1'k_2'} \tau_{ik_1'}\tau_{jk_2'}\pi_{k_1'k_2'}^{g_{ij}}(1-\pi_{k_1'k_2'})^{(1-g_{ij})}} \end{aligned} \tag{5.14}$$

5.3.2 M step: Maximizing the log likelihood

We will find the partial derivatives for each parameter of $\boldsymbol{\theta} : \boldsymbol{\tau}, \boldsymbol{\mu}, \boldsymbol{\sigma}$, and $\boldsymbol{\pi}$.

The parameters $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ only appear in 5.11, which is the same as in EM for Gaussian mixtures. This means the partial derivative is exactly the same as in the classical EM algorithm for Gaussian mixtures, se we find the same expressions:

Maximizing μ:

$$\hat{\mu_k} = \frac{1}{\sum_i Q_i(k)} \sum_i Q_i(k) x_i \tag{5.15}$$

Maximizing σ:

$$\hat{\sigma_k^2} = \frac{1}{\sum_i Q_i(k)} \sum_i Q_i(k)(x_i - \mu_k)^2 \tag{5.16}$$

For τ and π, we need to consider 5.12 too, so we find the derivative by hand.

Maximizing π: For this parameter, only Equation 5.12 appears in the derivative. First let us look at the derivative of $\pi_{k_1k_2}^{g_{ij}}(1-\pi_{k_1k_2})^{(1-g_{ij})}$, as it will be useful just after. This expression is just a way of writing

$$\pi_{k_1k_2}^{g_{ij}}(1-\pi_{k_1k_2})^{(1-g_{ij})} = \begin{cases} \pi_{rsk_1k_2} & \text{if } g_{ij}=1 \\ 1-\pi_{k_1k_2} & \text{if } g_{ij}=0 \end{cases}$$

In this case-by-case basis, the derivative with respect to $\pi_{k_1k_2}$ is clear:

$$\frac{\partial}{\partial \pi_{k_1k_2}} \pi_{k_1k_2}^{g_{ij}}(1-\pi_{k_1k_2})^{(1-g_{ij})} = \begin{cases} 1 & \text{if } g_{ij}=1 \\ -1 & \text{if } g_{ij}=0 \end{cases} = (-1)^{(1-g_{ij})} \tag{5.17}$$

With this in mind, we proceed derivating Equation 5.12:

$$\begin{aligned}
&\frac{\partial}{\partial \pi_{k_1k_2}} \sum_{ij} \sum_{k_1k_2} Q_{ij}(k_1,k_2) \log \frac{\tau_{ik_1}\tau_{jk_2}\pi_{k_1k_2}^{g_{ij}}(1-\pi_{k_1k_2})^{(1-g_{ij})}}{Q_{ij}(k_1,k_2)} \\
&= \sum_{\substack{i\in V_{k_1} \\ j\in V_{k_2}}} Q_{ij}(k_1,k_2) \frac{\partial}{\partial \pi_{k_1k_2}} \log \frac{\tau_{ik_1}\tau_{jk_2}\pi_{k_1k_2}^{g_{ij}}(1-\pi_{k_1k_2})^{(1-g_{ij})}}{Q_{ij}(k_1,k_2)} && \text{The only parts of the sum where } \pi_{k_1k_2} \text{ appears are when } c_i=\chi_{k_1}, c_j=\chi_{k_2} \text{ and we have both } k_1 \text{ and } k_2 \\
&= \sum_{\substack{i\in V_{k_1} \\ j\in V_{k_2}}} \frac{Q_{ij}(k_1,k_2)}{\pi_{k_1k_2}^{g_{ij}}(1-\pi_{k_1k_2})^{(1-g_{ij})}} \frac{\partial}{\partial \pi_{k_1k_2}} \pi_{k_1k_2}^{g_{ij}}(1-\pi_{k_1k_2})^{(1-g_{ij})} && \text{Since } \tfrac{\partial}{\partial x}\log(ax) = \tfrac{\partial}{\partial x}(\log(a)+\log(x)) = 0 + \tfrac{\partial}{\partial x}\log x = 1/x \\
&= \sum_{\substack{i\in V_{k_1} \\ j\in V_{k_2}}} \frac{Q_{ij}(k_1,k_2)}{\pi_{k_1k_2}^{g_{ij}}(1-\pi_{k_1k_2})^{(1-g_{ij})}} (-1)^{(1-g_{ij})} && \text{from Equation 5.17} \\
&= \sum_{\substack{i\in V_{k_1} \\ j\in V_{k_2} \\ g_{ij}=1}} \frac{Q_{ij}(k_1,k_2)}{\pi_{k_1k_2}} - \sum_{\substack{i\in V_{k_1} \\ j\in V_{k_2} \\ g_{ij}=0}} \frac{Q_{ij}(k_1,k_2)}{1-\pi_{k_1k_2}} && \text{By separating the sum into those } ij \text{ where } g_{ij}=1 \text{ and those where } g_{ij}=0 \\
&= \frac{1}{\pi_{k_1k_2}} \sum_{\substack{i\in V_{k_1} \\ j\in V_{k_2} \\ g_{ij}=1}} Q_{ij}(k_1,k_2) - \frac{1}{1-\pi_{k_1k_2}} \sum_{\substack{i\in V_{k_1} \\ j\in V_{k_2} \\ g_{ij}=0}} Q_{ij}(k_1,k_2)
\end{aligned}$$

We set this to zero and solve to find the optimal value of $\pi_{rsk_1k_2}$. First we observe the following bit of algebra:[§]

[§] Step by step: $\frac{a}{x} = \frac{b}{1-x}$, cross product
$\implies a(1-x) = bx$
$\implies a - ax = bx$
$\implies a = bx + ax$
$\implies a = (a+b)x$
$\implies \frac{a}{a+b} = x$, if $a+b \neq 0$

$$\frac{1}{x}a - \frac{1}{1-x}b = 0 \implies x = \frac{a}{a+b} \text{ if } a+b \neq 0 \text{ and } ab \neq 0$$

It follows naturally that

$$\begin{aligned}\hat{\pi}_{k_1k_2} &= \frac{\sum_{\substack{i\in V_{k_1}, j\in V_{k_2}\\ g_{ij}=1}} Q_{ij}(k_1,k_2)}{\sum_{\substack{i\in V_{k_1}, j\in V_{k_2}\\ g_{ij}=1}} Q_{ij}(k_1,k_2) + \sum_{\substack{i\in V_{k_1}, j\in V_{k_2}\\ g_{ij}=0}} Q_{ij}(k_1,k_2)} \\ &= \frac{\sum_{\substack{i\in V_{k_1}, j\in V_{k_2}\\ g_{ij}=1}} Q_{ij}(k_1,k_2)}{\sum_{i\in V_{k_1}, j\in V_{k_2}} Q_{ij}(k_1,k_2)} \end{aligned} \tag{5.18}$$

Maximizing τ: for this parameter, Equations 5.11 and 5.12 appear in the derivative.

$$\begin{aligned}
&\frac{\partial}{\partial\tau_k}\sum_i\sum_k Q_i(k)\log\frac{\tau_{ik}N(x_i|\mu_k,\sigma_k^2)}{Q_i(k)} \\
&+\frac{\partial}{\partial\tau_k}\sum_{ij}\sum_{k_1k_2} Q_{ij}(k_1,k_2)\log\frac{\tau_{ik_1}\tau_{jk_2}\pi_{k_1k_2}^{g_{ij}}(1-\pi_{k_1k_2})^{(1-g_{ij})}}{Q_{ij}(k_1,k_2)} \\
&= \sum_{i\in V_k} Q_i(k)\frac{\partial}{\partial\tau_k}\log\frac{\tau_k N(x_i|\mu_k,\sigma_k^2)}{Q_i(K_i=k)} + \sum_{i,j\in V_k}\Bigg(&& \text{the parts of the sum where } k \text{ appears and } \tau_{ik}=\tau_k\neq 0 \\
&\quad Q_{ij}(k,k)\frac{\partial}{\partial\tau_k}\log\frac{\tau_k^2\pi_{kk}^{g_{ij}}(1-\pi_{kk})^{(1-g_{ij})}}{Q_{ij}(k,k)} && \text{A: when } k_1=k \text{ and } k_2=k \\
&\quad + \sum_{\substack{k_2\neq k\\ \chi_k=\chi_{k_2}}} Q_{ij}(k,k_2)\frac{\partial}{\partial\tau_k}\log\frac{\tau_k\tau_{k_2}\pi_{kk_2}^{g_{ij}}(1-\pi_{kk_2})^{(1-g_{ij})}}{Q_{ij}(k,k_2)} && \text{B: when } k_1=k, \chi_k=\chi_{k_2} \text{ but } k_2\neq k \\
&\quad + \sum_{\substack{k_1\neq k\\ \chi_{k_1}=\chi_k}} Q_{ij}(k_1,k)\frac{\partial}{\partial\tau_k}\log\frac{\tau_{k_1}\tau_k\pi_{k_1k}^{g_{ij}}(1-\pi_{k_1k})^{(1-g_{ij})}}{Q_{ij}(k_1,k)}\Bigg) && \text{C: when } k_1\neq k, \text{ but } \chi_{k_1}=\chi_k \text{ and } k_2=k \\
&= \sum_{i\in V_k}\frac{Q_i(k)}{\tau_k} + \sum_{i,j\in V_k}\Bigg(\underbrace{2\frac{Q_{ij}(k,k)}{\tau_k}}_{\text{A}} + \underbrace{\sum_{\substack{k_2\neq k\\ \chi_k=\chi_{k_2}}}\frac{Q_{ij}(k,k_2)}{\tau_k}}_{\text{B}} + \underbrace{\sum_{\substack{k_1\neq k\\ \chi_{k_1}=\chi_k}}\frac{Q_{ij}(k_1,k)}{\tau_k}}_{\text{C}}\Bigg) && \text{Here again, we use } \tfrac{\partial}{\partial x}\log(ax)=1/x \\
&= \frac{1}{\tau_k}\sum_{i\in V_k} Q_i(k) + \frac{1}{\tau_k}\sum_{i,j\in V_k}\left(2Q_{ij}(k,k) + \sum_{\substack{k_2\neq k\\ \chi_k=\chi_{k_2}}} Q_{ij}(k,k_2) + \sum_{\substack{k_1\neq k\\ \chi_{k_1}=\chi_k}} Q_{ij}(k_1,k)\right) \\
&= \frac{1}{\tau_k}\left(\sum_{i\in V_k} Q_i(k) + \sum_{i,j\in V_k}\left(\sum_{\substack{k_1\\ \chi_k=\chi_{k_1}}} Q_{ij}(k_1,k) + \sum_{\substack{k_2\\ \chi_k=\chi_{k_2}}} Q_{ij}(k,k_2)\right)\right) \\
&= \frac{1}{\tau_k}\left(\sum_{i\in V_k} Q_i(k) + \sum_{i,j\in V_k}\sum_{\substack{k'\\ \chi_k=\chi_{k'}}}\left(Q_{ij}(k',k)+Q_{ij}(k,k')\right)\right)
\end{aligned}$$

We need to construct the Lagrangian to ensure that Equation 5.4 is verified:

$$\mathcal{L}(\tau_k) = (\dots) + \beta \left(\sum_{\substack{k' \\ \chi_k = \chi_{k'}}} (\tau_{k'} - 1) \right)$$

where β is the Lagrange multiplier. The derivative for τ_k is

$$\frac{\partial \mathcal{L}}{\partial \tau_k} = \frac{1}{\tau_k} \left(\sum_{i \in V_k} Q_i(k) + \sum_{i,j \in V_k} \sum_{\substack{k' \\ \chi_k = \chi_{k'}}} \left(Q_{ij}(k', k) + Q_{ij}(k, k') \right) \right) + \beta$$

If we set it to 0 to maximize τ_k, we get

$$\hat{\tau_k} = -\frac{1}{\beta} \left(\sum_{i \in V_k} Q_i(k) + \sum_{i,j \in V_k} \sum_{\substack{k' \\ \chi_k = \chi_{k'}}} \left(Q_{ij}(k', k) + Q_{ij}(k, k') \right) \right) \tag{5.19}$$

We can set β by solving $\sum_{\substack{k'=1 \\ \chi_k = \chi_{k'}}}^{K} \hat{\tau_{k'}} = 1$

$$\implies -\beta = \sum_k \left(\sum_{i \in V_k} Q_i(k) + \sum_{i,j \in V_k} \sum_{k'} \left(Q_{ij}(k', k) + Q_{ij}(k, k') \right) \right)$$

where the sums $\sum_k$ are only over the k sharing the same category χ_k

$$\implies -\beta = \sum_k \sum_{i \in V_k} Q_i(k) + \sum_k \sum_{i,j \in V_k} \sum_{k_1} Q_{ij}(k_1, k) + \sum_k \sum_{ij} \sum_{k_2} Q_{ij}(k, k_2)$$

$$\implies -\beta = \sum_{i \in V_k} 1 + \sum_{i,j \in V_k} 1 + \sum_{i,j \in V_k} 1$$

since Q_i and Q_{ij} are distributions over the different values of k sharing the same category χ_k

$$\implies \beta = -2|V_k|^2 - |V_k| \tag{5.20}$$

where $|V_k|$ is the number of vertices in category χ_k

5.3.3 *Putting it all together: pseudocode of the EM algorithm*

The resulting algorithm for finding the MLE of the parameters $\boldsymbol{\theta}$ is given in Algorithm 4. We will now discuss its performance and usefulness.

5.4 *Discussion*

The EM algorithm presented in Algorithm 4 performs very well. The guarantee that the likelihood increases at every iteration still holds, and generally the it converges quite quickly. A representative convergance curve for a small example is shown in Figure 5.2. The update steps are not disproportionately slower than for the greedy algorithm of the previous chapter. The mixtures of Gaussians are correctly detected.

Algorithm 4: Pseudocode for the adapted EM algorithm

Choose an initial setting for $\boldsymbol{\theta}$.
repeat
  E step:
  for $i \in V, k \leftarrow 1, \ldots, K$ **do**
    $Q_i(k) \leftarrow \frac{\tau_{ik} N(x_i|\mu_k,\sigma_k^2)}{\sum_{k'} \tau_{ik'} N(x_i|\mu_{k'},\sigma_{k'}^2)}$, as in Equation 5.13
  for $(i,j) \in V \times V, k_1 \leftarrow 1, \ldots, K, k_2 \leftarrow 1, \ldots, K$ **do**
    $Q_{ij}(k_1,k_2) \leftarrow \frac{\tau_{ik_1}\tau_{jk_2}\pi_{k_1k_2}^{g_{ij}}(1-\pi_{k_1k_2})^{(1-g_{ij})}}{\sum_{k_1'k_2'}\tau_{ik_1'}\tau_{jk_2'}\pi_{k_1'k_2'}^{g_{ij}}(1-\pi_{k_1'k_2'})^{(1-g_{ij})}}$, as in Equation 5.14
  M step:
  for $k \leftarrow 1, \ldots, K$ **do**
    $\mu_k \leftarrow \frac{1}{\sum_i Q_i(k)} \sum_i Q_i(k) x_i$, as in Equation 5.15
    $\sigma_k^2 \leftarrow \frac{1}{\sum_i Q_i(k)} \sum_i Q_i(k)(x_i - \mu_k)^2$, as in Equation 5.16
    $\tau_k \leftarrow \frac{1}{2|V_k|^2+|V_k|} \left(\sum_{i \in V_k} Q_i(k) + \sum_{i,j \in V_k} \sum_{\substack{k' \\ \chi_k = \chi_{k'}}} \left(Q_{ij}(k',k) + Q_{ij}(k,k') \right) \right)$, as in Equation 5.19 and Equation 5.20
  for $k_1 \leftarrow 1, \ldots, K, k_2 \leftarrow 1, \ldots, K$ **do**
    $\pi_{k_1k_2} \leftarrow \frac{\sum_{i \in V_{k_1}, j \in V_{k_2}} Q_{ij}(k_1,k_2)}{g_{ij}=1}$, as in Equation 5.18
until $\boldsymbol{\theta}$ or $\ell(\boldsymbol{\theta})$ converges

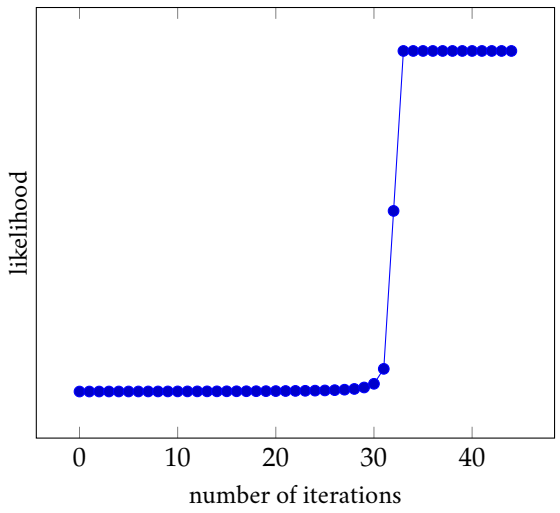


Figure 5.2: Convergence graph of our EM algorithm on a small example of 200 vertices.

There is however one major drawback of this approach, which is why I will not spend too much time discussing it. In simplifying the model to make it work with EM, we simplified it too much. Consider the following example in Figure 5.3, where the name of the vertices represent their category (either "a", "b" or "c") and their numerical value (either 1 or 0). When we run the EM algorithm to find the 6 underlying populations (two per category), we would like to get the inter-population edge densities π shown in Figure 5.5, where the name of the lines and columns refer to the populations of the vertex with the same name. Since there is only one vertex in each population, the inter-populations densities should be either 1 (we see an edge) or 0 (we don't see an edge). We expect the π matrix to look like the adjacency matrix.

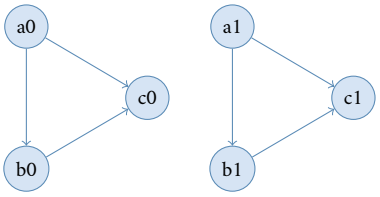


Figure 5.3: Small example of multidimensional network.

	a0	b0	c0	a1	b1	c1
a0	0	100	100	0	0	0
b0	0	0	100	0	0	0
c0	0	0	0	0	0	0
a1	0	0	0	0	100	100
b1	0	0	0	0	0	100
c1	0	0	0	0	0	0

Figure 5.4: expected π matrix (values in percent.).

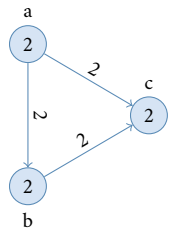
However this is not what we see. What we see instead is the following:

	a0	b0	c0	a1	b1	c1
a0	0	50	50	0	50	50
b0	0	0	50	0	0	50
c0	0	0	0	0	0	0
a1	0	50	50	0	50	50
b1	0	0	50	0	0	50
c1	0	0	0	0	0	0

Figure 5.5: actual π matrix (values in percent.).

This is not the expected matrix, but it is not too bad as it still captures some of the structure of the graph. However, this matrix does not differenciate between vertices using the numerical attribute. π_{a0b0} is the same as π_{a1b0} which is the same as π_{a1b1}, and so on. In fact, we could have gotten all of the information in this matrix by building the cuboid on the categorical attribute, and building the stochastic blockmodel. This can also be seen in the expression of $\hat{\pi}_{kk'}$ in Equation 5.18.

In general, the output for π of our EM algorithm is always the same probabilities as the blockmodel of the categorical cuboid —shown in Figure 5.6—, repeated a number of times, and scaled according to τ_k. This means that, although the algorithm is functionnal, the results do not help in the search for informative cuboids with numerical attributes. We can get the same information by building the categorical cuboid, and running EM for gaussian mixtures on the numerical attribute.



	a	b	c
a	0	50	50
b	0	0	50
c	0	0	0

Figure 5.6: Cuboid for our small example and the π matrix of its stochastic cuboid

In conclusion, even though our algorithm is correct, it is useless in the context of finding informative cuboids with numerical attributes. This realization was a major setback in the work for this thesis, and a lot of care and effort was put into the mathematical derivation of the algorithm.

6

A clustering approach for building informative cuboids

In this chapter, we will present a final approach to find informative cuboids with numerical attributes. This approach is based on the k-medoids clustering technique. Unfortunately, the idea for this approach came very late in the thesis' work, and so I did not get the time to investigate it in depth. However, the preliminary results are very promising, so I decided to include it in my thesis.

Intuitively, a cuboid is considered informative if it gives a succinct view of the graph, where "vertices which behave the same way are grouped together". What we mean by the vertex behavior is both the different attributes it has and what type of connections it has to other vertices. Let's say, for example, that we are working on a dataset from a music-focused social network. Users create their profile, where they state their favorite genre of music, and can become friends with other users. Let's image that, in our dataset, most fans of rap between 12 and 20 years old are friends only with other users fitting that same description. Older fans of rap, on the other hand, are friends with all kinds of people and not just other fans of rap. An informative cuboid on the (favorite genre, age) attributes would be able to group together the rap fans between 12 and 20 years, and show their unique behavior.

This suggest a conception of building informative cuboids as a *clustering problem*: We want to find clusters of vertices which share a similar "connective behavior". In this chapter, I will present a definition of "connective behavior" in the context of multidimensional networks with numerical attributes. I will then present a way to find these clusters using k-medoids. The preliminary results are presented. Finally, I discuss the limitations and possible improvements on this approach.

6.1 Describing the connective behavior of a vertex

We wish to cluster together vertices which have the same "connective behavior". Let's consider our musical social network example from before, and come up with three different members for our network:

1. Alice is 17 and likes rap music. She has 4 friends in the network. One is Bob, who is described later. The other 3 all like rap and have age 16, 17 and 19 respectively.

2. Bob is 40 has 7 friends in the network. One of them is Alice. Of the remaining six, three like country music and have age 40, 45 and 51. The three

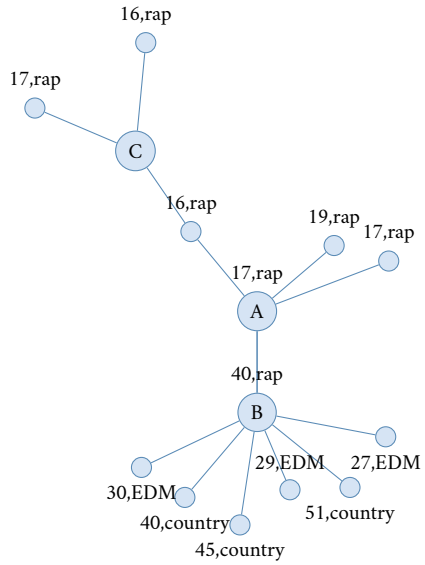


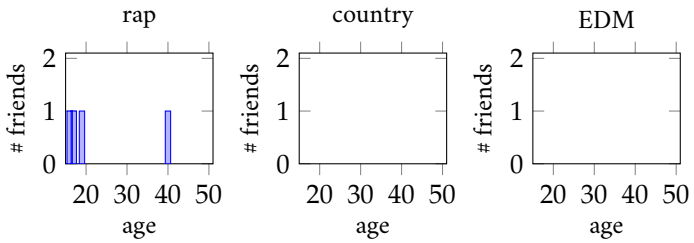
Figure 6.1: A small part of our example musical social media. Alice is represented by vertex A, Bob by vertex B and Charlie by vertex C.

remaining like EDM and have age 27, 29 and 30.

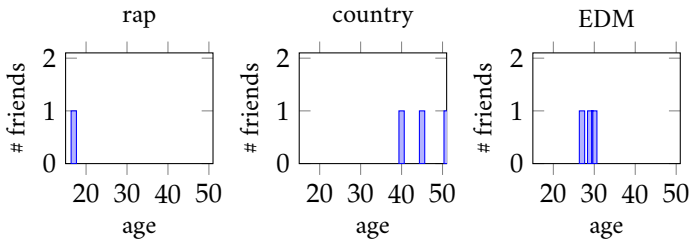
3. Charlie likes rap music. She has 3 friends in the network. They all like rap music and have age 17, 16 and 16 respectively.

We show this in graph form in Figure 6.1. We say that Alice and Charlie have a similar connective behavior, because they both have a small number of friends who are all young rap fans. Bob, on the other hand, has a very different connective behavior. He has many more friends, in different age groups and who like different kinds of music.

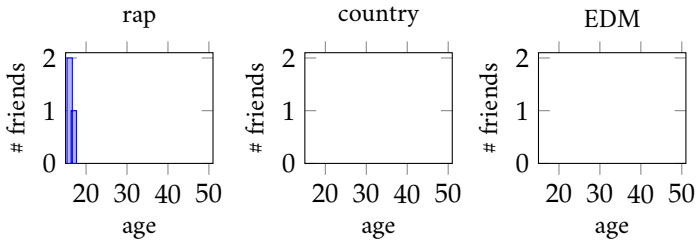
We can visualize the connective structure of a member of this network by drawing the histogram of the age of their friends, grouped by favorite genre of music. Here they are shown for Alice:



For Bob:



And for Charlie:



Users who share the same connective behavior will have similar histograms, i.e. they will have a similar distribution of the age of their friends, separated by favorite music genre. We will use this idea of *vector of distributions* as our formal notion of connective behavior: Given a setting with n different values for the categorical attribute, the connective behavior of a vertex v is a vector of n distributions. For each category c, the distribution is given by the numerical attribute of all vertex u in category c who has an edge to v*. We call this vector the *edge-distribution vector* of vertex i, and denote it by $\mathcal{V}_i$.

The proposed approach to build informative cuboids of multidimensional networks with numerical attributes is to group together the vertices sharing the same category, then cluster each group into k sub-groups, according to the similarity of the edge-distribution vectors $\mathcal{V}_i$ of the vertices in the cluster.

To build these clusters, we need a notion of distance between two edge-distribution vectors. The distance metric is presented in the next section.

* In the case of directed graphs, we have to count the incoming edges and the outgoing edges separately, so we get a vector of length $2n$ instead of n.

6.2 A distance metric for edge-distribution vectors

Suppose that we have a distance metric for distributions, $d : \mathcal{D} \times \mathcal{D} \longrightarrow \mathbb{R}$. Then, we can easily build a distance metric for vectors of distributions $d' : \mathcal{D}^n \times \mathcal{D}^n \longrightarrow \mathbb{R}$ by taking the sum of the distances of the elements of the vector, paired together:

$$d'(\boldsymbol{v}, \boldsymbol{w}) = \sum_{i=1}^{n} d(v_i, w_i)$$

This is exactly what we will use as the distance metric between two edge-distribution vectors. For the distance metric between two individual distributions, we will use Earth Mover's Distance (EMD, also called Wasserstein's metric [19]). The intuition for this distance (as well as the reason for its name) is the following:

[19]: "A metric for distributions with applications to image databases", Rubner, Tomasi, and Guibas, 1998

Consider the two distributions as two piles of dirt over a region D. The EMD distance between the two distributions is the minimal amount of effort needed to move the dirt of the first distribution, so that the pile has exactly the same shape as the second pile of dirt. For example, consider the distribution for the age of friends who like rap of Alice and Charlie, and imagine them as piles of dirt. In order to turn Alice's pile of dirt into Charlie's pile of dirt, she would have to move the stack of height 1 which is at position "age = 40" to the position "age=16", requiring a total effort of 1 * (40-16) = 24. This is their EMD distance.

EMD has the limitation that it is not a true metric if the two histograms are normalized — i.e. if the total amount of dirt is the same for the two piles. In our case, since vertices can have a different number of edges, the total amount of dirt varies from vector to vector. There is however an adaptation of EMD, called $\widehat{EMD}$ [20], which is a true metric even if the piles of dirt do not have equal mass. This adaptation adds a penalty for any mass added or lost. There is a linear-time algorithm to calculate this metric (also in [20]) which makes it a great candidate for our purposes. The distance metric for two edge-distributions vector is thus given by

[20]: "A Linear Time Histogram Metric for Improved SIFT Matching", Pele and Werman, 2008

$$d'(\mathcal{V}_a, \mathcal{V}_b) = \sum_{i=1}^{n} \widehat{EMD}(\mathcal{V}_{ai}, \mathcal{V}_{bi})$$

6.3 k-medoids clustering on edge-distribution vectors

k-medoids [21] is a variant of the k-means algorithm. Both algorithms are used to cluster a datasets into groups, by minimizing the sum of the distances from a "center point" of the cluster to every point in the cluster. In the case of k-means, this center point is the *mean* of the cluster — i.e. the point in space which minimizes the sum of distances. In the case of k-medoids this center is the *medoid* of the cluster — i.e. the element *from the dataset* which has the minimal sum of distances to every other point in the cluster.

[21]: "Clustering by Means of Medoids", Kaufmann and Rousseeuw, 1987

k-medoids is useful when we do not have a straightforward way of calculating the mean of a cluster, or when using a l_1 norm. Both are the case in our setting: Calculating since $\widehat{EMD}$ has no simple closed-form expression, calculating the mean with regards to this distance metric is not trivial, and the distance metric is l_1 since it is a simple sum of the distance for each

of the n dimensions. The pseudocode for the clustering algorithm is given in Algorithm 5.

Algorithm 5: k-medoids clustering algorithm for edge-distribution vectors

Input: A set of edge-distribution vectors S
Input: A set of k initial medoids $M = \{m_1, \ldots, m_k\} \subset S$
Output: k medoids which cluster S based on the d' distance.
while the total cost decreases **do**
    initialize k empty clusters $S_1, \ldots, S_k$
    for each $v \in S$ **do**
        find the medoid $m_i \in M$ for which $d'(m_i, v)$ is minimal
        add v to its cluster $S_i \leftarrow S_i \cup v$
    for each medoid m_i **do**
        update the medoid $m_i \leftarrow \operatorname{argmin}_{v \in S} \sum_{v' \in S_i} d'(v, v')$

Given a multidimensional network $\mathcal{N} = (V, E, A)$, the procedure to build an informative cuboid on categorical a attribute C and a numerical attribute N with this clustering technique is the following:

1. Build the categorical cuboid on attribute C;
2. For every condensed vertex $[v]$ in this cuboid, divide the vertices $u \in [v]$ according to the clusters given by Algorithm 5. This gives a new partition of the vertices;
3. Build the cuboid using this partition of the vertices.

Each aggregated vertex in the resulting cuboid can is described by a category and a representative element (its medoid). It groups together the vertices of that category which behave like the medoid, in the sense that their edge-distribution vectors are similar.

6.4 Preliminary results

To assess the effectiveness of this approach, I generated a graph with 500 vertices for each of the 4 scenarios given in subsection 4.3.2. I ran the k-medoids procedure to build the cuboid on these graphs. Then, I calculated the info score for the generated cuboids.

Remember that scenarios 1 and 2 had very different connective behaviors for every population sharing the same category. Scenarios 3 and 4, on the other hand, had much more similar behavior for the different populations. We expect our algorithm to perform better in 1 than in 3, and better in 2 than in 4, since the edge-distribution vectors will be more different between one cluster and the other.

Remember also that scenarios 1 and 3 had well-separated Gaussian mixtures, while scenarios 2 and 4 had overlapping Gaussian mixtures. We would expect our algorithm to perform better in scenario 1 than scenario 2, and better in 3 than 4, again because the edge-distribution vectors will be more different between clusters.

The results are shown in Table 6.1. We can see that our prediction that the results would be better for scenario 1 than 2 and better for 3 than 4 were correct. There is a net increase from one to the other. However, our prediction

that 1 would be better than 3 (and 2 better than 4) is completely wrong. One explanation for this might be that scenarios 1 and 2 have more chance of having vertices with no edges, which are impossible to cluster correctly using our approach.

	scenario			
	1	2	3	4
ideal	23,582	24,710	26,631	26,018
k-medoids	24,018	28,690	26,574	26,030
difference	463	3980	-57	12

Table 6.1: surprisal scores for the cuboids obtained by running the k-medoids algorithm on graphs of 500 vertices generated from our 4 scenarios. A smaller value is better. Remark that the algorithm overfitted the data, as we can see for example in scenario 3 where the score is ever better than the ideal score.

We can also see that the resulting models overfit the data, as we get better scores than the ideal scores. This is probably partly the fault of the size of our dataset: If we had more vertices, the dataset would look more like the model and have a (comparatively) higher ideal score.

We will now compare, for every scenario, the generating model with the model found by our algorithm. This will give us more insight into the performance of this approach.

6.4.1 Scenario 1

The comparison between the generating model and the model learned by our algorithm for scenario 1 is shown in Figure 6.2. We show the distributions of the generating model of the scenario in dotted lines, and a Gaussian fitted on the values of the corresponding cluster found by our algorithm in full lines. We also show the π matrix of the generating model, and the π matrix of the stochastic blockmodel built on our cuboid.

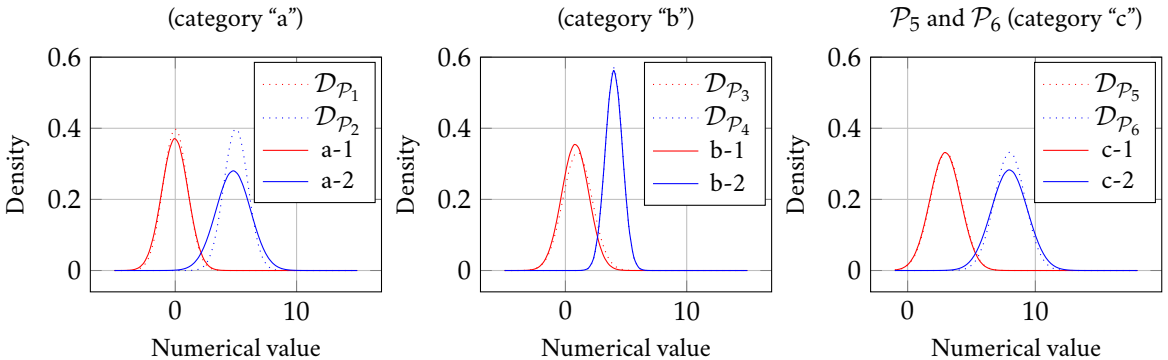
We can see that the distributions and edge-density matrices are nearly exactly the same. Out of the 4 scenarios, this is the one for which the generating and generated model match the best, which is what we had expected for our experiment.

6.4.2 Scenario 2

The comparison for scenario 2 is shown in Figure 6.3. As we can see, just as was reflected in the score, the cuboid is much less informative of the generating model. The two clusters for a seem to roughly correspond to population 1 and 2, but the other populations are mixed together. This is maybe caused by an unlucky initialization of the k-medoids algorithm.

6.4.3 Scenario 3

The comparison for scenario 3 is shown in Figure 6.4. While the distribution of the numerical attribute for every cluster is not what we expected to see, the edge-density matrix is quite similar. This is probably because the edge densities for populations sharing the same categorical attribute are so similar: A vertex of $\mathcal{P}_1$ can easily deviate enough from its edge-density probabilities to have a edge-distribution vector which looks like that of a vertex of $\mathcal{P}_2$. This is a plausible explanation, since we have also seen that the cuboid overfits the dataset very much in this scenario.



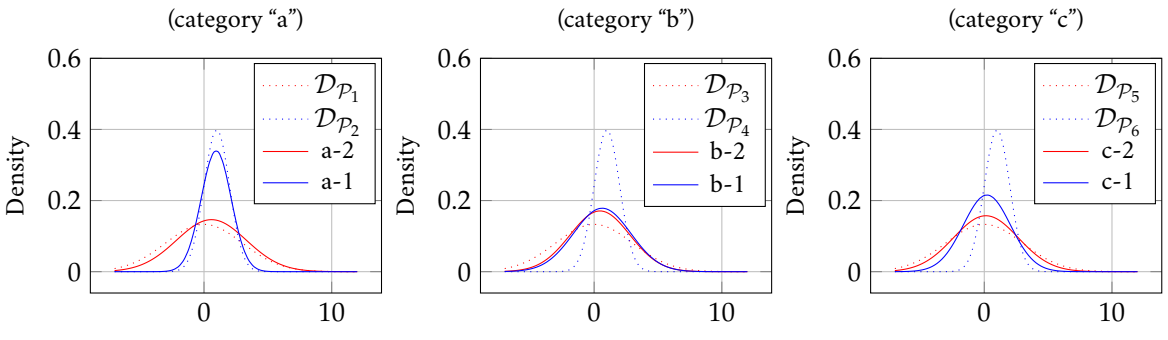
π for the generating model:

	1	2	3	4	5	6
1	60.0	90.0	00.0	80.0	70.0	0.0
2	10.0	10.0	70.0	00.0	00.0	80.0
3	00.0	00.0	80.0	00.0	00.0	0.0
4	00.0	00.0	00.0	30.0	00.0	0.0
5	80.0	60.0	00.0	00.0	10.0	50.0
6	00.0	00.0	80.0	50.0	50.0	0.0

π of the cuboid built on the clusters:

	a-1	a-2	b-1	b-2	c-1	c-2
a-1	55.6	85.7	0.0	81.0	71.4	1.5
a-2	9.9	11.5	57.3	4.1	1.9	70.8
b-1	0.0	0.0	90.7	0.5	0.0	0.0
b-2	0.0	0.0	0.4	30.7	0.0	0.0
c-1	78.2	55.9	0.0	0.0	11.7	50.2
c-2	2.0	1.8	77.4	48.1	46.3	0.8

Figure 6.2: Comparison between the generating model and the cuboid for scenario 1. The graphs show in dotted lines the generating distributions, and in full strokes the distributions in each cluster. The matrices show the probabilities in ‰.



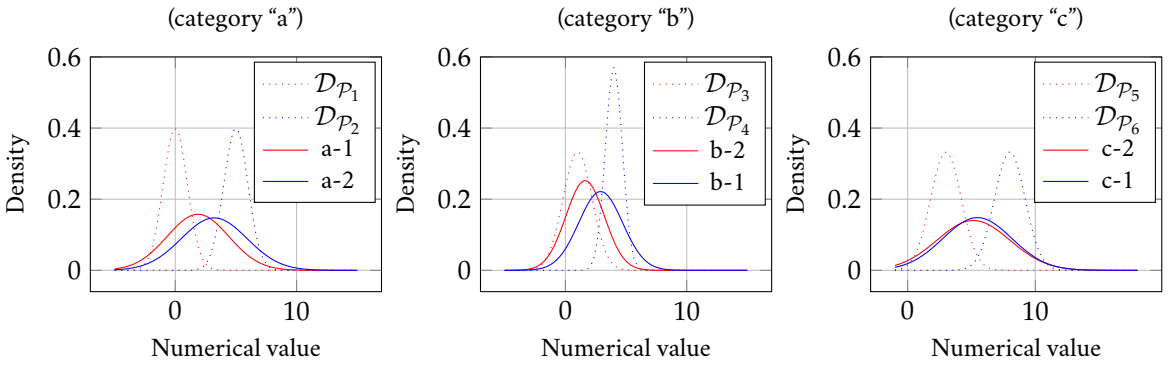
π for the generating model:

	1	2	3	4	5	6
1	60.0	90.0	00.0	80.0	70.0	0.0
2	10.0	10.0	70.0	00.0	00.0	80.0
3	00.0	00.0	80.0	00.0	00.0	0.0
4	00.0	00.0	00.0	30.0	00.0	0.0
5	80.0	60.0	00.0	00.0	10.0	50.0
6	00.0	00.0	80.0	50.0	50.0	0.0

π of the cuboid built on the clusters:

	a-2	a-1	b-2	b-1	c-2	c-1
a-2	47.7	61.8	39.0	30.7	37.0	43.4
a-1	15.8	16.0	41.4	43.4	18.0	40.8
b-2	0.0	0.0	27.2	42.9	0.0	0.0
b-1	0.0	0.0	26.9	44.4	0.0	0.0
c-2	50.1	46.2	15.0	17.7	24.9	29.1
c-1	25.2	23.4	53.2	58.5	34.1	31.1

Figure 6.3: Comparison of the generating model and the cuboid found for scenario 2. The graphs show in dotted lines the generating distributions, and in full strokes the distributions in each cluster. The matrices show the probabilities in ‰.



π for the generating model:

	1	2	3	4	5	6
1	40.0	50.0	60.0	40.0	0.0	0.0
2	50.0	40.0	70.0	40.0	0.0	0.0
3	0.0	0.0	10.0	20.0	40.0	40.0
4	0.0	0.0	20.0	10.0	30.0	30.0
5	70.0	50.0	0.0	0.0	30.0	20.0
6	60.0	70.0	0.0	0.0	20.0	20.0

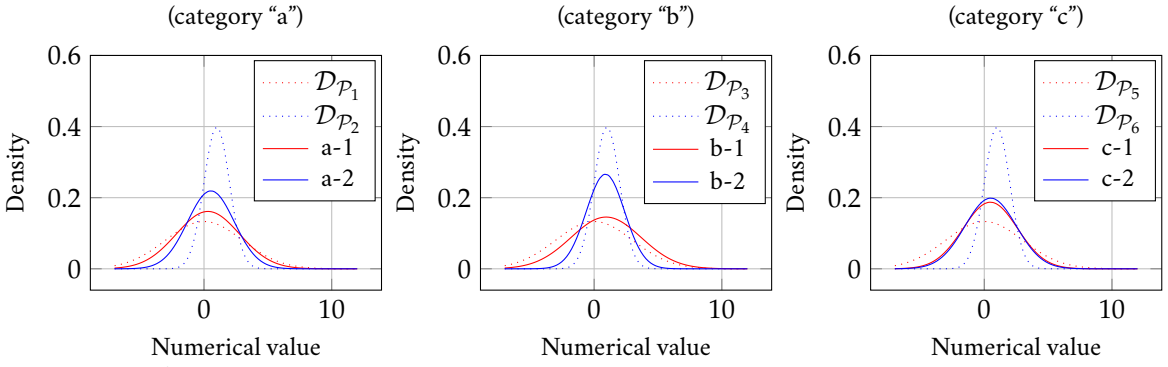
π of the cuboid built on the clusters:

	a-1	a-2	b-2	b-1	c-2	c-1
a-1	49.2	40.5	55.5	46.9	0.0	0.0
a-2	34.3	24.4	77.9	53.7	0.0	0.0
b-2	0.0	0.0	11.7	16.6	47.3	44.7
b-1	0.0	0.0	16.3	15.8	25.1	23.2
c-2	45.9	34.8	0.0	0.0	31.3	19.2
c-1	68.4	65.6	0.0	0.0	24.9	21.5

Figure 6.4: Comparison between the generating model and the cuboid for scenario 3. The graphs show in dotted lines the generating distributions, and in full strokes the distributions in each cluster. The matrices show the probabilities in ‰.

6.4.4 *Scenario 4*

The comparison is shown in Figure 6.5. The results are similar to those of scenario 3, but there seem to be less misclassified vertices.



π for the generating model:

	1	2	3	4	5	6
1	40.0	50.0	60.0	40.0	0.0	0.0
2	50.0	40.0	70.0	40.0	0.0	0.0
3	0.0	0.0	10.0	20.0	40.0	40.0
4	0.0	0.0	20.0	10.0	30.0	30.0
5	70.0	50.0	0.0	0.0	30.0	20.0
6	60.0	70.0	0.0	0.0	20.0	20.0

π of the cuboid built on the clusters:

	a-1	a-2	b-1	b-2	c-1	c-2
a-1	45.7	44.3	53.2	30.3	0.0	0.0
a-2	48.3	46.4	88.3	45.9	0.0	0.0
b-1	0.0	0.0	13.7	16.1	35.9	38.5
b-2	0.0	0.0	13.9	14.9	28.6	24.3
c-1	50.3	45.3	0.0	0.0	24.9	21.1
c-2	78.1	63.6	0.0	0.0	26.1	22.4

Figure 6.5: Comparison of the generating model and the cuboid found for scenario 4. The graphs show in dotted lines the generating distributions, and in full strokes the distributions in each cluster. The matrices show the probabilities in ‰.

6.5 *Discussion*

As we have seen, the preliminary results are very promising. The description of vertices as edge-distribution vectors seem to capture what we intuitively understood as the behavior of a vertex of the network. The algorithm is

relatively simple to implement, and simple to understand. The resulting description of the overall behavior of the network in terms of categories with representative individuals (the medoids) could lead to real insight into the structure of the graph, which is ultimately the goal of graph mining.

One drawback of this approach is that the k-medoids procedure disregards completely the numerical attribute of the vertices it is clustering. It only considers the numerical value of the end points of its edges, through the edge-distribution vector. This can be compensated by adding a new term to the distance between two vertices, which is simply the distance between their numerical attributes. The distance between a vertex i and a vertex j would then be $|x_i - x_j| + d'(\mathcal{V}_i, \mathcal{V}_j)$. This would push the vertices with similar numerical attributes closer together, and would probably improve the performance in cases like scenario 2 (seen in subsection 6.4.2).

A second drawback is the time complexity of the k-medoids algorithm I presented in Algorithm 5. To calculate the new medoid of a cluster, we calculate, for every point in the dataset, the sum of distances to every other point of the cluster. Since there are k clusters, and given n elements, this has time complexity $O(k(n-k)^2)$. There is a better implementation of k-medoids which has $O(nk)$ time complexity, given in [22]. This is as good as one can expect from this kind of technique.

[22]: "A simple and fast algorithm for K-medoids clustering", Park and Jun, 2009

A third drawback is intrinsic to the k-medoids technique. If vertices in our graph have very few edges on average, it might be that no single vertex in the graph captures the complete behavior of its population. In that case, k-medoids will not be able to find a good medoid to cluster the whole population. This problem would be avoided if we used k-means, since the center for clusters would no longer be restricted to elements of the dataset. In a k-means approach, the center could have a edge-distribution vector with fractional number of edges — e.g. the center of a cluster could be 0.25 friends who like rap in the 20 to 25 age group, and 0.7 friends who like country in the 40 to 50 age group.

To use k-means, we would need a procedure to find the mean of a set of distributions for the $\widehat{EMD}$ distance. There are some techniques to find an approximation of the mean in the literature — see [23] for a polynomial time approximation, or [24] for an approach using neural networks — but the simplest seems to be from [25]. Their approach is to use a type of gradient descent called Weiszfeld's algorithm, which was first presented in [26] in 1936 (see [27] for a more modern approach). Any of these techniques greatly increases the time complexity of the clustering algorithm. Furthermore, I have tried implementing the Weiszfeld approach myself, but got some erratic behavior in the clustering loop. Nonetheless, I think these would be very interesting topics for further research, especially since it seems to bring together ideas from many different fields of research — e.g. image databases, lexicon induction, Bayesian analysis, ...

[23]: "FPTAS for Minimizing the Earth Mover's Distance Under Rigid Transformations and Related Problems", Ding and Xu, 2017

[24]: "Earth Mover's Distance Minimization for Unsupervised Bilingual Lexicon Induction", Zhang, Liu, Luan, *et al.*, 2017

[25]: "Robust and Scalable Bayes via a Median of Subset Posterior Measures", Minsker, Srivastava, Lin, *et al.*, 2014

[26]: "Sur un problème de minimum dans l'espace", Weiszfeld, 1936

[27]: "Weiszfeld's Method", Beck and Sabach, 2015

7
Conclusion

The objective of this thesis was to explore different ways of finding informative cuboids with numerical attributes. We first presented a new connection between the theory of graph cubes and stochastic blockmodels. We detailed the exact nature of this connection, and used this to construct a measure of informativeness for cuboids. We also presented a new experimental setup to test different approaches to building informative cuboids with synthetic datasets.

We explored first a simple binning approach, which can give good results when there is a clear sparation between numerical attributes of different populations. We explored a second approach based on EM, which unfortunately appears to be a dead-end. Finally, we explored a third approach, based on a new description of vertices based on the distribution of the end point of its edges. We presented some exciting first results for this approach, and then discussed different areas of improvement.

There are many directions in which to extend this work. The measure of informativeness of cuboids could be extended to the case of multigraphs, or to graphs with heterogenous vertex types. Different search procedures could be considered to find informative binnings of the numerical attributes, for example with specific constraints in a CP framework. The idea of clustering to build cuboids could be further explored by looking at different clustering techniques, and especially graph clustering techniques. We could imagine a cuboid built with a community-detection clustering algorithm, for example.

Bibliography

[1] E. R. Tufte, *Beautiful evidence*. Cheshire: Graphics press, 2006, 213 pp., OCLC: ocm70203994, ISBN: 978-0-9613921-7-8.

[2] P. Zhao, X. Li, D. Xin, and J. Han, "Graph cube: On warehousing and OLAP multidimensional networks", 2011. [Online]. Available: https://research.google.com/pubs/pub37657.html (visited on 04/30/2018).

[3] A. Ghrab, O. Romero, S. Skhiri, A. Vaisman, and E. Zimányi, "A framework for building OLAP cubes on graphs", in *Advances in Databases and Information Systems*, ser. Lecture Notes in Computer Science, Springer, Cham, Sep. 8, 2015, pp. 92–105, ISBN: 978-3-319-23134-1 978-3-319-23135-8. DOI: 10.1007/978-3-319-23135-8_7. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-23135-8_7 (visited on 08/19/2018).

[4] F. Demesmaeker, "Graph cube mining", PhD thesis, École polytechnique de Louvain, Université catholique de Louvain, 2017.

[5] F. Demesmaeker, A. Ghrab, S. Nijssen, and S. Skhiri, "Discovering interesting patterns in large graph cubes", *IEEE International Conference on Big Data (Big Data)*, pp. 3322–3331, 2017. [Online]. Available: https://research.euranova.eu/wp-content/uploads/Discovering-interesting-patterns-in-large-graph-cubes.pdf (visited on 08/19/2018).

[6] A. Datta and H. Thomas, "The cube data model: A conceptual model and algebra for on-line analytical processing in data warehouses", *Decision Support Systems*, vol. 27, no. 3, pp. 289–301, 1999, ISSN: 0167-9236. DOI: https://doi.org/10.1016/S0167-9236(99)00052-4.

[7] S. Chaudhuri and U. Dayal, "An overview of data warehousing and OLAP technology", *SIGMOD Rec.*, vol. 26, no. 1, pp. 65–74, Mar. 1997, ISSN: 0163-5808. DOI: 10.1145/248603.248616. [Online]. Available: http://doi.acm.org/10.1145/248603.248616 (visited on 04/30/2018).

[8] J. Vreeken and N. Tatti, "Interesting patterns", in *Frequent Pattern Mining*, C. C. Aggarwal and J. Han, Eds., Springer, 2014, pp. 105–134, ISBN: 978-3-319-07820-5. DOI: 10.1007/978-3-319-07821-2_5.

[9] J. Dougherty, R. Kohavi, and M. Sahami, "Supervised and unsupervised discretization of continuous features", in *Machine Learning Proceedings 1995*, Elsevier, 1995, pp. 194–202.

[10] H. C. White, S. A. Boorman, and R. L. Breiger, "Social structure from multiple networks. i. blockmodels of roles and positions", *American Journal of Sociology*, vol. 81, no. 4, pp. 730–780, Jan. 1, 1976, ISSN: 0002-9602. DOI: 10.1086/226141. [Online]. Available: https://www.journals.uchicago.edu/doi/10.1086/226141 (visited on 08/05/2018).

[11] P. W. Holland, K. B. Laskey, and S. Leinhardt, "Stochastic blockmodels: First steps", *Social Networks*, vol. 5, no. 2, pp. 109–137, Jun. 1, 1983, ISSN: 0378-8733. DOI: 10.1016/0378-8733(83)90021-7. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0378873383900217 (visited on 03/21/2018).

[12] E. Abbe and C. Sandon, "Community detection in general stochastic block models: Fundamental limits and efficient recovery algorithms", *arXiv:1503.00609 [cs, math]*, Mar. 2, 2015. arXiv: 1503.00609. [Online]. Available: http://arxiv.org/abs/1503.00609 (visited on 08/05/2018).

[13] D. S. Jones, *Elementary information theory*, ser. Oxford applied mathematics and computing science series. Oxford: Clarendon, 1979, 182 pp., ISBN: 978-0-19-859637-0.

[14] F. M. Harper and J. A. Konstan, "The MovieLens datasets: History and context", *ACM Trans. Interact. Intell. Syst.*, vol. 5, no. 4, 19:1–19:19, Dec. 2015, ISSN: 2160-6455. DOI: 10.1145/2827872. [Online]. Available: http://doi.acm.org/10.1145/2827872 (visited on 08/15/2018).

[15] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008, ISBN: 978-0-521-86571-5.

[16] G. Casella and R. L. Berger, *Statistical inference*, 2nd ed. Delhi: Cengage, 2002, 660 pp., ISBN: 978-81-315-0394-2.

[17] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm", *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 39, no. 1, pp. 1–38, 1977, ISSN: 0035-9246. [Online]. Available: https://www.jstor.org/stable/2984875 (visited on 08/16/2018).

[18] C. M. Bishop, *Pattern recognition and machine learning*, ser. Information science and statistics. New York: Springer, 2006, 738 pp., ISBN: 978-0-387-31073-2.

[19] Y. Rubner, C. Tomasi, and L. J. Guibas, "A metric for distributions with applications to image databases", in *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, Jan. 1998, pp. 59–66. DOI: 10.1109/ICCV.1998.710701.

[20] O. Pele and M. Werman, "A linear time histogram metric for improved SIFT matching", in *Computer Vision – ECCV 2008*, ser. Lecture Notes in Computer Science, vol. 5304, Springer Berlin Heidelberg, Jan. 1, 2008, pp. 495–508, ISBN: 978-3-540-88689-1.

[21] L. Kaufmann and P. Rousseeuw, "Clustering by means of medoids", *Data Analysis based on the L1-Norm and Related Methods*, pp. 405–416, Jan. 1, 1987.

[22] H.-S. Park and C.-H. Jun, "A simple and fast algorithm for k-medoids clustering", *Expert Systems with Applications*, vol. 36, no. 2, pp. 3336–3341, Mar. 1, 2009, ISSN: 0957-4174. DOI: 10.1016/j.eswa.2008.01.039. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S095741740800081X (visited on 08/19/2018).

[23] H. Ding and J. Xu, "FPTAS for minimizing the earth mover's distance under rigid transformations and related problems", *Algorithmica*, vol. 78, no. 3, pp. 741–770, Jul. 1, 2017, ISSN: 0178-4617, 1432-0541. DOI: 10.1007/s00453-016-0173-4. [Online]. Available: https://link-springer-com.proxy.bib.ucl.ac.be:2443/article/10.1007/s00453-016-0173-4 (visited on 08/18/2018).

[24] M. Zhang, Y. Liu, H. Luan, and M. Sun, "Earth mover's distance minimization for unsupervised bilingual lexicon induction", in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017, pp. 1934–1945.

[25] S. Minsker, S. Srivastava, L. Lin, and D. B. Dunson, "Robust and scalable bayes via a median of subset posterior measures", *arXiv:1403.2660 [cs, math, stat]*, Mar. 11, 2014. arXiv: 1403.2660. [Online]. Available: http://arxiv.org/abs/1403.2660 (visited on 08/19/2018).

[26] E. Weiszfeld, "Sur un problème de minimum dans l'espace", *Tohoku Mathematical Journal, First Series*, vol. 42, pp. 274–280, 1936, ISSN: 0040-8735, 1881-2015. [Online]. Available: https://www.jstage.jst.go.jp/article/tmj1911/42/0/42_0_274/_article (visited on 08/19/2018).

[27] A. Beck and S. Sabach, "Weiszfeld's method: Old and new results", *Journal of Optimization Theory and Applications*, vol. 164, no. 1, p. 1, Jan. 1, 2015, ISSN: 0022-3239. DOI: 10.1007/s10957-014-0586-7.

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve **www.uclouvain.be/epl**