MINERANDO PADRÕES REAIS EM TENSORES INCERTOS

LUCAS JOSÉ CARNEIRO MACIEL

MINERANDO PADRÕES REAIS EM TENSORES INCERTOS

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Loïc Cerf
Coorientador: Vinícius dos Santos

Belo Horizonte
Março de 2018

LUCAS JOSÉ CARNEIRO MACIEL

MINING REAL PATTERNS IN UNCERTAIN TENSORS

Dissertation presented to the Graduate Program in Computer Science of the Federal University of Minas Gerais in partial fulfillment of the requirements for the degree of Master in Computer Science.

Advisor: Loïc Cerf
Co-Advisor: Vinícius dos Santos

Belo Horizonte
March 2018

Maciel, Lucas José Carneiro

C152m Mining Real Patterns in Uncertain Tensors / Lucas José Carneiro Maciel. — Belo Horizonte, 2018
xxiv, 58 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais - Departamento de Ciência da Computação.

Orientador: Loïc Pascal Gilles Cerf
Corientador: Vinícius Fernandes dos Santos

1. Computação — Teses. 2. Mineração de Dados (Computação). 3. Programação Linear. 4. Modelo de Regressão. 5. Reconhecimento de Padrões. I. Orientador. II. Coorientador. III. Título.

CDU 519.6*73(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Mining real patterns in uncertain tensors

LUCAS JOSÉ CARNEIRO MACIEL

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. LOIC PASCAL GILLES CERF - Orientador
Departamento Ciência da Computação - UFMG

PROF. VINICIUS FERNANDES DOS SANTOS - Coorientador
Departamento de Ciência da Computação - UFMG

PROF. FLAVIO VINICIUS DINIZ DE FIGUEIREDO
Departamento Ciência da Computação - UFMG

PROF. THIAGO FERREIRA DE NORONHA
Departamento de Ciência da Computação - UFMG

PROF. ROBSON LEONARDO FERREIRA CORDEIRO
Instituto de Ciências Matemáticas e de Computação - USP

Belo Horizonte, 4 de abril de 2018.

Acknowledgments

This dissertation was achieve after so much effort and the support of those who were present in my life on the past few years. First, I would like to thank God for giving me the wisdom and health enough to get here, and Holy Mary for guiding me.

I would like to express my very great appreciation to Professor Loïc Cerf and Professor Vinícius dos Santos, my research supervisors, for their patient guidance, encouragement, useful critiques and countless reviews of all work we have done together, including this dissertation. I would also like to thank Professor Thiago Noronha for his advice and assistance since my undergraduate course. My grateful thanks are also extended to all Professors of the Computer Science Department of UFMG, for the precious classes that help me to build all my knowledge in Computer/Data Science. Special thanks to PPGCC and Google for the financial support during the Master's program.

Finally, I wish to thank my parents, Cláudio and Gláucia, for their support and encouragement throughout my study, João and Ariane for all their help and my wife, Stella, her affection, love and support gave me the strenght to conclude this research work.

This study was financed in part by the Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG) through the project APQ-04224-16 of Multilateral Collaboration FAPEMIG-CNRS.

"Probability is expectation founded upon partial knowledge. A perfect acquaintance with all the circumstances affecting the occurrence of an event would change expectation into certainty, and leave neither room nor demand for a theory of probabilities."

(George Boole)

Resumo

Tensores incertos codificam o quanto são satisfeitos predicados n-ários. Por exemplo, o tempo que usuários gastam em diferentes sites da web em cada semana podem ser transformados em graus de interesse que os usuários (1ª dimensão) tem pelos sites (2ª dimensão) durante as semanas (3ª dimensão). No resultante tensor incerto tridimensional, sub-tensores que são grandes e densos frequentemente são interessantes para um analista. Eles representam usuários que têm mostrado muito interesse nos mesmos sites durante as mesmas semanas. Mirkin and Kramarenko propuseram um *modelo disjuntivo de box cluster* (*disjunctive box cluster model*), que é um modelo de regressão onde os padrões são variáveis explicativas dos valores no tensor incerto. Nesta dissertação, duas abordagens são propostas de acordo com tal modelo. Fragmentos dos padrões de interesse são primeiramente providos por algoritmos completos. Estes fragmentos são então crescidos, na primeira abordagem, usando um procedimento conhecido como hill-climbing. Em cada iteração deste procedimento, um problema de programação linear inteira é resolvido para encontrar um padrão maior. Já na segunda abordagem, os fragmentos são hierárquicamente aglomerados. Em ambas abordagens, pré-processamentos são propostos para acelerar a subsequente computação. Finalmente, uma técnica de regressão gradual, *forward selection*, seleciona entre os padrões descobertos, um subconjunto não redundante que melhor descreve o tensor sem causar overfit. Experimentos em ambos tensores sintéticos e reais mostram que as propostas descobrem padrões de alta qualidade em tensores incertos e superam o estado da arte quando aplicados a tensores 0/1, um caso específico.

Palavras-chave: Mineração de padrões, tensores incertos, disjunctive box cluster model, hill-climbing, programação linear inteira, aglomeração hierárquica, algoritmos de seleção.

Abstract

Uncertain tensors encode to what extent n-ary predicates are satisfied. For instance, the times users spent on different websites week after week can be turned into degrees of interest of the users (1^{st} dimension) for the sites (2^{nd} dimension) during the weeks (3^{rd} dimension). In the resulting 3-way uncertain tensor, sub-tensors that are both large and dense are often interesting to an analyst. They are users who all showed much interest for the same sites during the same weeks. Mirkin and Kramarenko proposed the *disjunctive box cluster model*, a regression model where such patterns are explanatory variables for the values in the uncertain tensor. In this dissertation, two approaches are proposed to fit a disjunctive box cluster model to an uncertain tensor. A complete algorithm first provides fragments of the desired patterns. In the first approach, a hill-climbing procedure individually grows them. At every iteration of that procedure, integer linear programming is used to compute the larger pattern. In the second approach, the input fragments are hierarchically agglomerated. In both cases, greedy pre-processes are proposed to speed up the subsequent computation. Finally, a stepwise regression technique, the *forward selection*, chooses among the discovered patterns a non-redundant subset that fits, but does not overfit, the tensor. Experiments on both synthetic and real-world tensors show the proposals discovers high-quality patterns in uncertain tensors and outperforms state-of-the-art approaches when applied to 0/1 tensors, a special case.

Keywords: Pattern mining, uncertain tensor, disjunctive box cluster model, hill-climbing, integer linear programming, hierarchical agglomeration, forward selection

List of Figures

List of Tables

Contents

Chapter 1

Introduction

Suppose an analyst wants to correlate the types of chocolate Brazilian consumers like with the regions they live in. She can have a table (or matrix) where each row represents one Brazilian region, each column represents one type of chocolate and every cell contains the amount of chocolate of the type in column that were sold in the region in row. To use the sale data as a proxy to how much Brazilian consumers in different regions like the different types of chocolate, the analyst can turn the numerical data at her disposal into discrete values. To do so, she can define intervals of sales and map these intervals to "hates", "likes a little", "likes", and "loves". However, discretizing is losing information: after such a process there is no way to recover the original numerical value and two values in a same interval cannot be distinguished anymore. To avoid that, the analyst needs a bijection between the sale data and $[0, 1]$, where 0 stands for "absolutely hating", 1 for "absolutely loving", and values in between express, in a continuous way, all degrees of "liking". She can normalize the data and choose any function, for example a logistic function, i. e., defining its two parameters (what is "moderately liking" and how sharp the transition from "hating" to "loving"), and use it to transform every numerical value in the matrix into a value in $[0, 1]$. The resulting matrix, with values in $[0, 1]$, is said *uncertain* because it quantifies the certainty the analyst has that the consumers in the different regions like the different types of chocolates.

Table 1.1 shows an uncertain matrix our analyst may end up with. Large sub-matrices with values close to 1 are of interest: they indicate subsets of regions (the rows of a sub-matrix) that all like a subset of the types of chocolate (the columns of a sub-matrix). Table 1.1 emphasizes two such interesting sub-matrices. The first one (darker cells) means that consumers in the `South`, `Southeast` and `Northeast` regions like `milky`, `sweet` and `white` chocolates. Its area is 9 and its

	bitter	crunchy	milky	semisweet	sweet	white
South	0.29	0.05	0.89	0.04	0.94	0.89
Southeast	0.69	0.76	0.83	0.84	0.98	0.87
North	0.82	0.96	0.15	0.49	0.87	0.51
Northeast	0.52	0.09	0.86	0.07	0.94	0.48
West-Center	0.91	0.81	0.05	0.73	0.33	0.39

Table 1.1: Chocolate sales uncertain matrix, with two patterns highlighted in gray and dark gray.

density (i. e., the average value in it) is $\frac{7.68}{9} \approx 0.85$. The second one (grayed cells) is {Southeast,North,West-Center} × {bitter,crunchy, semisweet,sweet} of area 12 and density $\frac{9.19}{12} \approx 0.77$.

This dissertation deals with the discovery of such large and dense sub-matrices. In fact, it considers a broader setting: the *uncertain tensor*. The *patterns* of interest therefore become large and dense sub-tensors. Not only every discovered pattern should be of interest but, as a whole, these patterns should make a good summary of the uncertain tensor, i. e., a small number of patterns should be returned (hence, a possible human interpretation), that altogether allow an approximate reconstruction of the uncertain tensor.

The classical data mining tasks deal with the discovery of either *global patterns* or *local patterns*. A clustering, a classification function or a regression function are examples of global patterns, i. e., of patterns that organize an entire dataset (by partitioning the objects in the case of clustering, by explaining a target attribute as a function of the other attributes in the case of classification/regression). Frequent patterns such as frequent itemsets and association rules, which support the discovery of correlations between multiple attributes in a transactional dataset, are examples of local patterns, i. e., of patterns that emphasize interesting portions of the dataset. Searching for a few local patterns that altogether sum up a dataset is building a global pattern from local patterns. This dissertation therefore faces difficulties coming from both world.

In the local pattern world, time complexity is a significant issue. The number of candidate patterns (here sub-tensors) is exponential in the size of the dataset. That is why enumerating all possible patterns and testing whether they are interesting is intractable. The uncertain matrix in Table 1.1 being 5×6, it contains $(2^5-1)\times(2^6-1) = 1,953$ non-empty sub-matrices because any non-empty subset of row associated with any non-empty subset of columns defines a sub-matrix. In a slightly larger sub-matrix,

of size 20×20, the number of candidate patterns exceeds one trillion. Yet, by defining the patterns of interest in a rather constrained way, there are algorithms that manage to list them all in large datasets. Those constrained definitions have downside. First of all, the number of patterns may still be exponential in the size of the dataset. It is not even uncommon to compute a collection of patterns that is larger than the dataset. Then, the definitions mainly (if not only) deal with the portion of the dataset that the pattern covers. That entails much redundancy of information in the returned collection of patterns, nothing preventing the patterns from overlapping a lot. Finally, real-life datasets are noisy. In our example, deriving "liking degrees" from the sale data is imperfect. For instance, discounts influence the sales but not the tastes. The varying costs of the involved commodities too. When combined with the necessity of strong constraints (e. g., a minimal density, minimal sizes, etc.) on the patterns, to get reasonable running times, the noise effectively prevents *complete* algorithms (i. e., algorithms listing *all* patterns satisfying the chosen definition) from discovering large patterns. Only fragments of them are discovered.

In the global pattern world, the optimal pattern may be defined but the related algorithms, e. g., to cluster objects or search for a classification/regression function, rely on heuristics and do not guarantee they will find this optimum. Again, the reason is the exponential time complexity of the exact problems. The quality of a global pattern does not only depend on how accurately it fits the data. It must be simple too. For instance, defining one cluster per object minimizes k-means' objective function but is not a good clustering. Taking into account all explanatory variables to build a regression function minimizes the residuals but is not a good model of the data either. Not only a complicated global pattern is harder to interpret but it captures the noise in the data.

The idea of building global models from local patterns, as in this dissertation, is not new. In particular, associative classification is about building a classifier from a set of association rules concluding on a class attribute. Processing such local patterns rather than or in complement with the original dataset is postponing the use of lossy heuristics, what hopefully results in a global model that is more trustworthy. However, that also means dealing with the problems listed earlier: many small and redundant local patterns must be turned into a simple model. In this dissertation, the global model being a set of patterns, the smaller the cardinality of this set, the simpler the model. The discovery of the two patterns in Table 1.1 followed the local-to-global approach. They were grown from the small and redundant patterns that Table 1.2 lists and that a complete algorithm returned. Table 1.3 highlights those smaller patterns in the uncertain matrix. Because they carry redundant information (i. e., overlap a lot),

$\mathcal{F}$	**Pattern**	**Area**	**Density**
F_1	{Southeast, West-Center} × {crunchy, semisweet}	4	0.7850
F_2	{Southeast, North, West-Center} × {bitter, crunchy}	6	0.8250
F_3	{South, Southeast, Northeast} × {milky, sweet}	6	0.9067
F_4	{Southeast, West-Center} × {bitter, semisweet}	4	0.7925
F_5	{Southeast, North} × {bitter, crunchy, sweet}	6	0.8467
F_6	{South, Southeast} × {milky, sweet, white}	6	0.9000

Table 1.2: Six small and redundant patterns, their their areas and their densities.

	bitter	crunchy	milky	semisweet	sweet	white
South	0.29	0.05	0.89	0.04	0.94	0.89
Southeast	0.69	0.76	0.83	0.84	0.98	0.87
North	0.82	0.96	0.15	0.49	0.87	0.51
Northeast	0.52	0.09	0.86	0.07	0.94	0.48
West-Center	0.91	0.81	0.05	0.73	0.33	0.39

	bitter	crunchy	milky	semisweet	sweet	white
South	0.29	0.05	0.89	0.04	0.94	0.89
Southeast	0.69	0.76	0.83	0.84	0.98	0.87
North	0.82	0.96	0.15	0.49	0.87	0.51
Northeast	0.52	0.09	0.86	0.07	0.94	0.48
West-Center	0.91	0.81	0.05	0.73	0.33	0.39

Table 1.3: Cells of the uncertain matrix that the six patterns in Table 1.2 cover. For clarity, the first three patterns are shown in the top matrix, whereas the last three patterns are shown in the bottom matrix. The gray scale represents the pattern density, darker meaning denser.

the patterns are shown three by three.

Since the patterns that complete algorithms can discover in a reasonable time are usually fragments of the desired patterns, the problem looks like a "jigsaw puzzle", whose pieces are overlapping. Several works [Sylvain Blachon et al., 2007; Gunjan K. Gupta and Ghosh, 1999; Hannu Toivonen et al., 1995; Wong and Li., 2008; Zhao and Zaki., 2005; Cerf et al., 2009b] tried to solve that puzzle when the tensor (usually, only matrices) is 0/1 and the pieces of the puzzle are sub-tensors full of 1. In that same context, the recent literature largely abandoned the local-to-global approach due to the exponential number of fragments to summarize and tried to directly and heuristically discover the global patterns in the tensor. That problem is known under the name *Boolean Tensor Factorization (BTF)* because the elements of the pat-

	bitter	crunchy	milky	semisweet	sweet	white
South	0	0	1	0	1	1
Southeast	1	1	1	1	1	1
North	1	1	1	0	1	1
Northeast	1	0	1	0	1	0
West-Center	1	1	0	1	0	0

Table 1.4: 0/1 matrix obtained by rounding every value in the uncertain matrix in Table 1.1 and two patterns (highlighted in gray and dark gray) discovered in this 0/1 matrix.

terns (its rows and the columns if the 0/1 tensor is actually a matrix) can be encoded in Boolean matrices (one per dimension of the dataset) whose outer product should approximately reconstruct the 0/1 tensor.

To the best of our knowledge, this dissertation details the first algorithms to summarize *uncertain* tensors with patterns. 0/1 tensors (and matrices) are a special case. As exemplified earlier with the chocolate sale data, being able to handle uncertain data allows to avoid a lossy pre-processing step in applications where the original data is numeric and the patterns of interest are sub-tensors with high (or low) values. In other terms, rounding to either 0 or 1 the values, in $[0, 1]$, of an uncertain tensor harms the ability to discover the pattern in the original dataset. For example, rounding every value in the uncertain matrix in Table 1.1 results in the 0/1 matrix in Table 1.4. Two large and dense patterns are discovered in that 0/1 matrix but they differ from those discovered in the uncertain matrix; Looking at the original data, they are worse.

In 2011, Mirkin and Kramarenko [2011] defined a regression model whose explanatory variables are patterns and the tensor (which is 0/1 in Mirkin and Kramarenko [2011]) is the target variable. More precisely, that model predicts that every value in the tensor is the density of the densest pattern covering it. In that framework, the best single pattern is the one with the largest product of its area with the square of the average value in it (i. e., the squared density). Mirkin and Kramarenko propose hill-climbing to search for patterns locally maximizing that objective function.

This dissertation proposes two algorithms to discover patterns that are candidate explanatory variables for Mirkin and Kramarenko's regression model. Figure 1.1 shows the proposed workflow to summarize *uncertain* tensors within that model. Starting from an uncertain tensor, a set of fragments are mined in a complete way

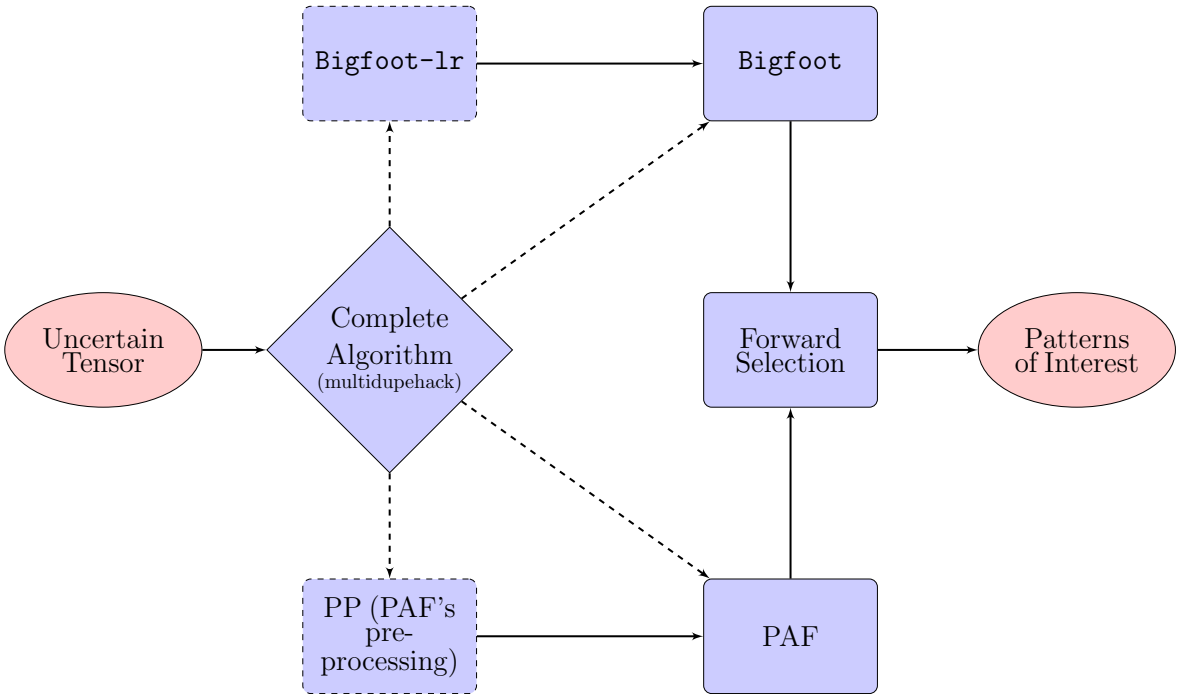


Figure 1.1: Proposed workflow for discovering patterns from an uncertain tensor. The dashed boxes are optional steps. There are two ways to discover patterns, with either `Bigfoot` or PAF.

(`multidupehack` [Cerf and Meira Jr., 2014] is strongly suggested). Next, there are two options of algorithms which this dissertation proposes, both of them including an optional step to reduce the set of fragments with pre-processing procedures. At end a stepwise regression algorithm is also proposed to select over the mined patterns, the explanatory variables for the model.

The first algorithm, `Bigfoot`[1], is the closest to Mirkin and Kramarenko's proposal. It relies as well on hill-climbing, but in a different parameter space, to grow every small pattern at its input into a pattern with a locally maximal area times density squared. The second algorithm, PAF[2], grows as well small patterns that a complete algorithm typically returns but does so by hierarchical agglomeration. Figure 1.2 shows the dendrogram resulting from the hierarchical agglomeration of the six small pattern, the leaves of the dendrogram, in Table 1.2. Contrary to `Bigfoot`, PAF can discover nested patterns, i. e., large patterns that contain smaller but denser patterns. Once the candidate patterns built, either by `Bigfoot` or by PAF, this dissertation proposes to use a stepwise regression technique, the *forward selection*, to compose the final model.

[1] `Bigfoot` is a recursive acronym. It stands for `Bigfoot` Is Growing Fragments Out Of Tensors. It did not exist in nature. We designed it.

[2] PAF stands for "Papas At Farm", which is, itself, an acronym for "PAF Agglomerates Patterns And Selects Agglomerates To Fit A Regression Model".

It selects a small number of candidate patterns that fit but do not overfit the uncertain tensor. Any of the two algorithms in addition to the stepwise regression technique can be used to summarize *uncertain* tensors.

After an overview of the techniques used in this dissertation, in Section 1.1, and a few basic definitions in Section 1.2, Chapter 2 presents Mirkin and Kramarenko's regression model and other related works. Chapter 3 presents the `Bigfoot` algorithm, detailing its hill-climbing in the pattern dimension space. `Bigfoot` solves an Integer Linear Programming problem (or its relaxation) at every iteration of the hill-climbing procedure. Chapter 4 deals with PAF. A similarity non-metric measure between patterns is defined and PAF's hierarchical agglomeration is detailed. A pre-process is proposed as well in that chapter. It upper-bounds PAF's space complexity, decreases its time requirements but does not affect much the quality of the patterns that are built. Chapter 5 details the forward selection of the final model: a small set of patterns, among those that `Bigfoot` or PAF discovers, that fits but not overfits the uncertain tensor. Experiments in Chapter 6 compare `Bigfoot` and PAF with state-of-art algorithms. Using real-world uncertain tensors, it shows as well that both approaches can quickly turn tens of thousands of small patterns, with much redundancy of information, into a few large patterns that summarize the uncertain tensor. Finally, Chapter 7 concludes the dissertation and discusses future work.

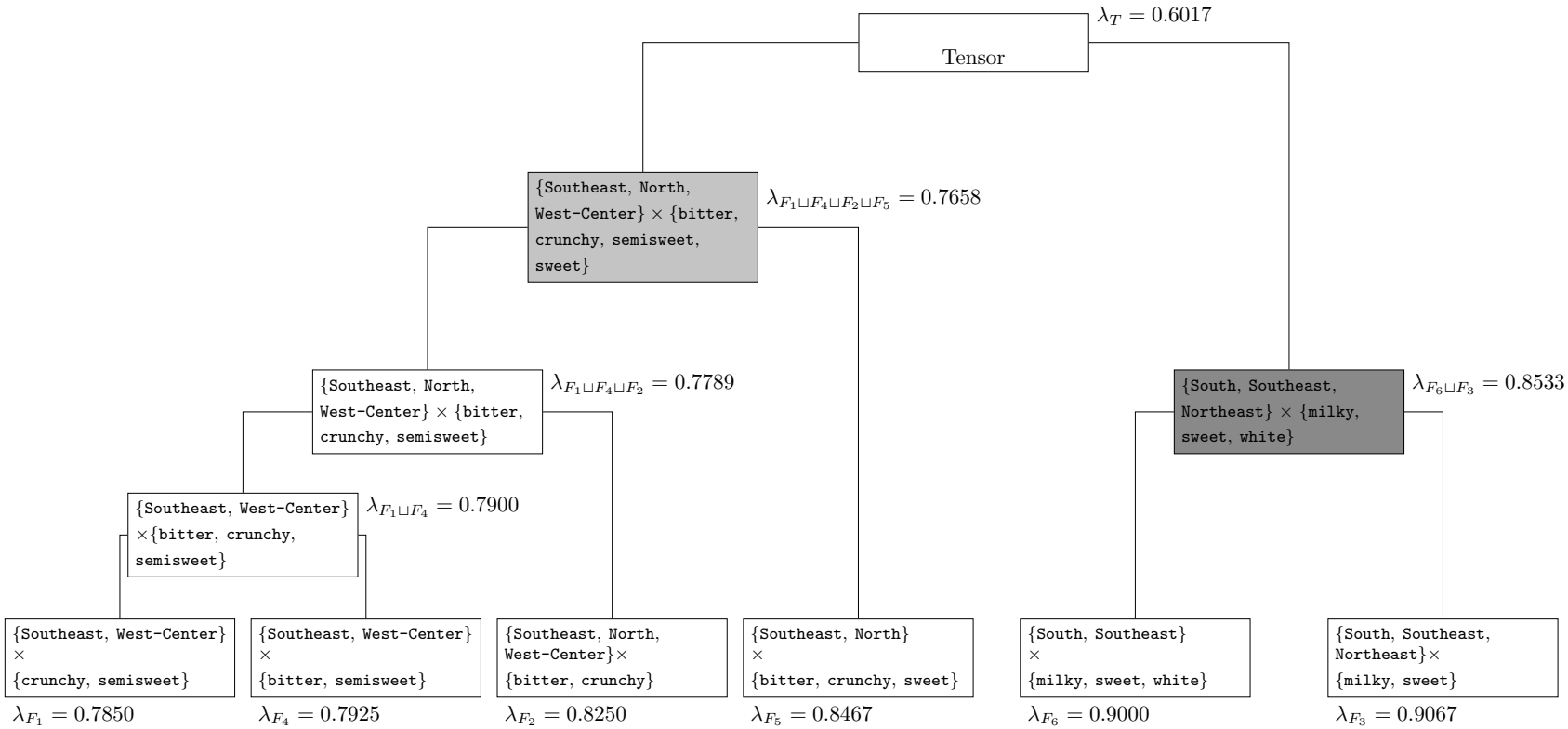


Figure 1.2: PAF dendrogram result. The two patterns of the chocolate sales tensor discovered by the proposals of this work are highlighted in gray cells.

1.1 Overview of the Techniques Used in this Dissertation

Integer programming aims to optimize a function under a set of constraints whose variables are integers. In an *Integer Linear Programming (ILP)* problem, the objective function and the constraints are linear [Schrijver, 1998]. Integer programming is NP-hard [Nemhauser and Wolsey, 1988]. An ILP problem can be seen as finding a point $x \in \mathbb{R}^n$ inside an convex hyper-polyhedron (given by the union of the all half-hyperspaces defined by the constraints) such that x maximizes the objective function. Many algorithms solve ILP problems [Erwin Abbink, 2017]. The most popular ones are the exact methods. The *cutting-plane* methods solve relaxations of linear programming problems whose variables can become real numbers. They create a new constraint to reduce the hyper-polyhedron into a smaller one until the optimal solution of the relaxation is also integer. *Constraint separation* is a generalization of the cutting-plane method for ILP problems with an exponential number of constraints, whereas the *column generation* method deals with ILP problems where the number of variables is exponential. There are methods that use the "divide and conquer" paradigm to explore the set of feasible solutions: branch-and-bound, cut-and-branch, branch-and-cut and branch-and-price. Those methods use lower and upper bounds to make significantly cuts in the solution space. The CPLEX optimizer [Meindl and Templ, 2013] is one of the faster solvers. It implements all those methods (and more) to quickly solve ILP problems. Despite the theoretical exponential time, CPLEX's computation takes almost polynomial time to solve any ILP instance [Bixby et al., 1988].

Hierarchical clustering is a data mining method to build a hierarchy of clusters, i. e., a dendrogram as in Figure 1.2. A cluster is simply a set of objects that are similar. The choice of the similarity measure therefore has a fundamental impact on the obtained clustering. There are two types of hierarchical clustering [Rokach and Maimon, 2005]. The hierarchical *agglomeration* is a bottom-up hierarchical clustering: every object is initially alone in a cluster, and clusters are greedily merged pairwise until only one cluster remains. On the contrary, the hierarchical *division* initially considers that all the objects are in one single cluster and, iteratively, the largest cluster is greedily split until there is one cluster per object. In this dissertation, PAF hierarchically *agglomerates* the small patterns at its input, i. e., the "objects" are these patterns.

1.2 Basic Definitions and Notations

Given $n \in \mathbb{N}$ *dimensions* (i. e., n finite sets, assumed disjoint without loss of generality) $D_1, \ldots, D_n$, an *uncertain tensor* T maps any n-tuple $t \in \prod_{i=1}^{n} D_i$ (where $\prod$ denotes the Cartesian product) to a value $T_t \in [0, 1]$, called *membership degree* of t. In this dissertation, a pattern in T is the Cartesian product of subsets of each of the n dimensions. Formally, a set of n-tuples $X \subseteq \prod_{i=1}^{n} D_i$ is a *pattern* if and only if there exists $X_1 \subseteq D_1, \ldots, X_n \subseteq D_n$ such that $X = \prod_{i=1}^{n} X_i$. Given two patterns X and Y, X is a *sub-pattern* of Y and Y is a *super-pattern* of X if and only if $X \subseteq Y$. Agglomerating $X = \prod_{i=1}^{n} X_i$ and $Y = \prod_{i=1}^{n} Y_i$ results in the smallest pattern including both X and Y, i. e., $\prod_{i=1}^{n} X_i \cup Y_i$. It is denoted $X \sqcup Y$.

Given an n-tuple $t \in \prod_{i=1}^{n} D_i$ and $i \in \{1, \ldots, n\}$, t_i denotes the i^{th} component of t, hence an element of D_i. Given a pattern $X \subseteq \prod_{i=1}^{n} D_i$ and an element $e \in D_i$, $\{t \in X \mid t_i = e\}$ is called a *slice* of X. For instance, if $n = 2$, the slices of a pattern are its rows and columns.

Taking the uncertain tensor in Table 1.1, $n = 2$ and the two dimensions are:

$$D_{\texttt{regions}} = \{\texttt{South}, \texttt{Southeast}, \texttt{North}, \texttt{Northeast}, \texttt{West-Center}\};$$

$$D_{\texttt{chocolates}} = \{\texttt{bitter}, \texttt{crunchy}, \texttt{milky}, \texttt{semisweet}, \texttt{sweet}, \texttt{white}\}.$$

The two patterns emphasized in Table 1.1 are:

$$P_1 = \{\texttt{South}, \texttt{Southeast}, \texttt{Northeast}\} \times \{\texttt{milky}, \texttt{sweet}, \texttt{white}\};$$

$$P_2 = \{\texttt{Southeast},\texttt{North},\texttt{West-Center}\} \times \{\texttt{bitter},\texttt{crunchy}, \texttt{semisweet},\texttt{sweet}\}.$$

The agglomeration of those two patterns is:

$$X_1 \sqcup X_2 = \{\texttt{South}, \texttt{Southeast}, \texttt{Northeast}, \texttt{North}, \texttt{West-Center}\} \times \{\texttt{bitter}, \texttt{crunchy}, \texttt{milky}, \texttt{semisweet}, \texttt{sweet}, \texttt{white}\}.$$

The slice $e = \texttt{South}$ of X_1 is $\{\texttt{South}\} \times \{\texttt{milky}, \texttt{sweet}, \texttt{white}\}$. Given the 2-tuple $t = (\texttt{South}, \texttt{milky})$, $t_1 = \texttt{South}$ and $t_2 = \texttt{milky}$.

Chapter 2

Related Work

The definition in Section 1.2 of a pattern is purely syntactical. To be semantically *relevant*, a pattern must contain a "large" number of tuples and their membership degrees must be "high". The data mining literature on pattern mining in 0/1 or uncertain tensors formalizes and fulfills those objectives in different ways.

2.1 Complete Algorithms

2.1.1 0/1 Matrix Mining

This work focuses on mining tensors. A few exceptions are made here for algorithms that only handle 0/1 matrices but technically relate to `Bigfoot`. `AC-Close` [Cheng et al., 2006] lists closed patterns with dense rows and dense columns (two lower-bounds are specified) by growing small patterns whose tuples all have membership degrees equal to 1: the frequent closed itemsets. Poernomo and Gopalkrishnan use ILP to output complete collections of patterns with upper-bounded numbers [Poernomo and Gopalkrishnan, 2007] or proportions [Poernomo and Gopalkrishnan, 2009] of tuples having null membership degrees.

2.1.2 0/1 Tensor Mining

Given a 0/1 tensor, several algorithms, notably Data-Peeler [Cerf et al., 2009a], list the closed patterns whose n-tuples all have membership degrees equal to 1. Ignatov et al. [2015] survey relaxations of that definition. RMiner [Spyropoulou et al., 2014] does not relax Data-Peeler's definition but generalizes it to a multi-relational context: 0/1 tensors sharing dimensions. To tolerate tuples with null membership de-

grees, A-RMiner [Spyropoulou and Bie, 2014] grows intermediary patterns that RMiner computes and that still only includes tuples with membership degrees equal to 1. Every output pattern is a minimal pattern that contains all RMiner's patterns sharing such a common fragment. Given prior beliefs, Spyropoulou et al. [2014] and Spyropoulou and Bie [2014] deem a pattern interesting if the ratio between its information content, which grows with its number of tuples, and its description length, which grows with its number of elements in all dimensions, is high.

2.1.3 Uncertain Tensor Mining

`multidupehack` [Cerf and Meira Jr., 2014] generalizes DATA-PEELER so that uncertain tensors can be mined: n bounds $(\epsilon_1, \ldots, \epsilon_n) \in \mathbb{R}_+^n$ specify how much the total membership degree of the n-tuples in every slice of a pattern can deviate from the sum that would be obtained if these membership degrees were all 1. Formally, a pattern $X = \prod_{i=1}^{n} X_i$ output by `multidupehack` satisfies $\forall i \in \{1, \ldots, n\}$, $\forall e \in X_i$, $\sum_{t \in X \text{ s.t. } t_i = e} (1 - T_t) \leq \epsilon_i$. `multidupehack`, like DATA-PEELER, enforces as well a closedness constraint, which discards all strict sub-patterns of a valid pattern, and handles additional relevance constraints that prune the search of the patterns satisfying them. The minimal size constraint — "involving at least $\gamma_i \in \mathbb{N}$ elements of D_i" — and the minimal area constraint — "having at least $\nu \in \mathbb{N}$ n-tuples" — are examples of constraints that force every output pattern to be large enough and that greatly shorten the run time. `multidupehack` actually computed the patterns in Table 1.2. They are, in the uncertain matrix in Table 1.1, all the patterns with at least two regions and two types of chocolate ($\gamma_1 = \gamma_2 = 2$) when the tolerance to noise is $\epsilon_1 = \epsilon_2 = 0.6$. A consequence of the definition of a pattern and of the chosen threshold is that every row and every column of a minimally-sized (i. e., 2×2) pattern has at least a $\frac{2-0.6}{2} = 0.7$ density.

To the best of our knowledge, DCE [Georgii et al., 2011] is the only other complete algorithm to mine uncertain tensors. Using synthetic datasets, Cerf and Meira Jr. [2014] show that DCE's definition of a valid pattern tends to catch patterns that go over the edges of the patterns that were planted, whereas `multidupehack`'s patterns do not, unless the tensor is very noisy. Moreover, `multidupehack` scales better than DCE. Yet, increasing its bounds $(\epsilon_1, \ldots, \epsilon_n)$ still exponentially influences the run time: given a reasonable time budget, `multidupehack` can only return many overlapping fragments of a large and noisy pattern to discover.

2.2 Heuristic Algorithms

2.2.1 Agglomerating Patterns

TRICLUSTER [Zhao and Zaki, 2005] can optionally post-process the patterns it mines in 3-way tensors. It can merge two overlapping patterns X and Y into $X \sqcup Y$. That happens when $\frac{|X \cup Y|}{|X \sqcup Y| - |X \cup Y|}$ is large enough. That process ignores the membership degrees. In contrast, ALPHA [Cerf et al., 2009b] takes into account the membership degrees of the n-tuples in $X \sqcup Y$ to define the similarity between the patterns X and Y in a 0/1 tensor T. That similarity is $\min_{i=1}^{n} \min_{e \in X_i \cup Y_i} \frac{\sum_{t \in X \sqcup Y \text{ s.t. } t_i = e} T_t}{|\{t \in X \sqcup Y \text{ s.t. } t_i = e\}|}$, i. e., the smallest density among those of the "slices" of $X \sqcup Y$. ALPHA hierarchically agglomerates patterns according to that similarity. It then selects a pattern in the dendrogram if it is distant from its parent and close to only having n-tuples with membership degrees equal to 1. A difference combines the two criteria. TRICLUSTER's and ALPHA's similarities lack theoretical grounds.

Wong and Li [2008] propose to hierarchically agglomerate patterns in 0/1 matrices. The distance between two patterns X and Y is a sum of normalized entropy in the two rectangular regions of $(X \sqcup Y) \setminus (X \cup Y)$, weighted by their respective areas. That computation sees the elements in one of the two dimensions as independent probabilistic events and the distance between two patterns depends on the choice of this dimension, i. e., the distance between X and Y is different if the 0/1 matrix is transposed. It therefore is unclear how the distance could be generalized to patterns in 0/1 tensors.

2.2.2 Directly Mining 0/1 Tensors

Several algorithms [Leenen et al., 1999; Miettinen, 2011; Bělohlávek et al., 2013; Erdös and Miettinen, 2013; Metzler and Miettinen, 2015; Park et al., 2017] approximately factorize n-way 0/1 tensors. To be easily interpreted as r patterns, the n factors of the rank-r CANDECOMP/PARAFAC (CP) decomposition are forced to be 0/1 too. More precisely, the Boolean rank-r CP decomposition of a tensor $T \in \{0,1\}^{D_1 \times \cdots \times D_n}$ aims to discover n matrices $A^1 \in \{0,1\}^{|D_1| \times r}$, …, $A^n \in \{0,1\}^{|D_n| \times r}$ that minimize $\|T - \max_{k=1}^{r} A^1_{:k} \otimes \cdots \otimes A^n_{:k}\|$, where $A^i_{:k}$ denotes the k^{th} column of A^i, $\otimes$ is the outer product, $\|.\|$ is the Frobenius norm [Dummit and Foote, 2003], and $\max_{k=1}^{r}$ returns a tensor where the membership degree of an n-tuple is 1 if it is 1 in at least one of r tensors, otherwise 0. In each of those r tensors, the pattern is the set of n-tuples with value 1, i. e., $A^1_{:k} \otimes A^2_{:k} \otimes A^3_{:k}$, where $k \in \{1, \ldots, r\}$ identifies the pattern. Figure 2.1 illustrates the CP decomposition of a 3-way 0/1 tensor T. The outer product of the

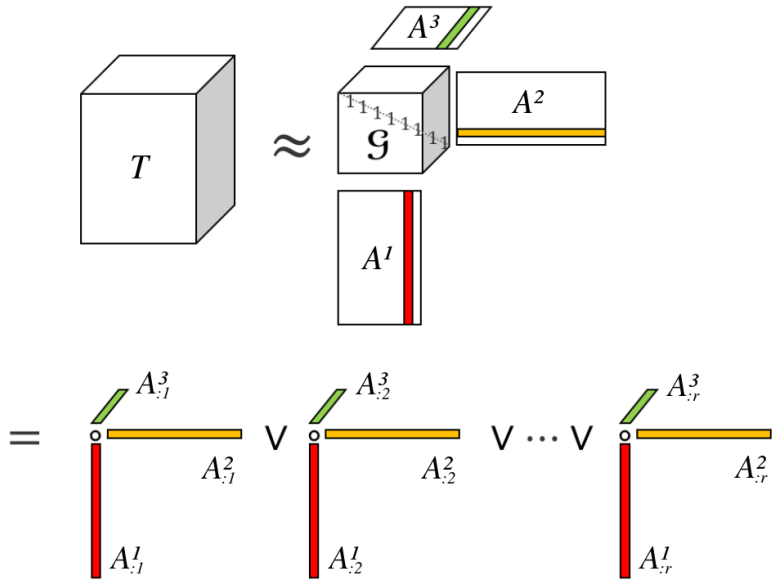


Figure 2.1: Boolean rank-r CP decomposition of a 3-way tensor T. T is decomposed into three factors, i. e., three 0/1 matrices A^1, A^2 and A^3.

	X_1	X_2
South	1	0
Southeast	1	1
North	1	1
Northeast	1	0
West-Center	0	1

	X_1	X_2
bitter	1	1
crunchy	0	1
milky	1	0
semisweet	0	1
sweet	1	0
white	0	1

Table 2.1: A rank-2 CP decomposition of the 0/1 matrix in Table 1.4. Each of the two factors relates to each of the two dimensions of the matrix. The outer product of the first column of each of the two factors gives the first pattern, $\{\texttt{South}, \texttt{Southest}, \texttt{North}, \texttt{Northest}\} \times \{\texttt{bitter}, \texttt{milky}, \texttt{sweet}\}$. Analogously, the second pattern is $\{\texttt{Southest}, \texttt{North}, \texttt{West} - \texttt{Center}\} \times \{\texttt{bitter}, \texttt{crunchy}, \texttt{semi} - \texttt{sweet}, \texttt{white}\}$.

three 0/1 factors is a 0/1 tensor that aims to be as close as possible (w.r.t. the Frobenius distance) to the original tensor T (top figure). Intuitively, that product plants in a null tensor, having the same sizes as T, r patterns, i. e., r (possibly overlapping) sub-tensors full of 1 (bottom figure). The k^{th} such pattern ($k \in \{1, \ldots, r\}$) is the outer product of the k^{th} column of each of the three factors. The r patterns minimizing the objective function are usually large and dense.

`BCP_ALS` [Miettinen, 2011] heuristically seeks them by Alternating Least Squares, a method that heuristically solves in quadratic time the non-convex optimization prob-

lem of the CP decomposition by alternatively fixing $n-1$ factors and computing the remaining one. DBTF [Park et al., 2017] distributes that method on the Spark framework and exhibits near-linear scalability with respect to the size of the 0/1 tensor, its density, the rank r (a parameter) of the factorization and the number of machines. Table 2.1 presents the two factors, with $r = 2$, that DBTF computes from the rounded matrix in Table 1.4.

Walk'n'Merge [Erdös and Miettinen, 2013] discovers patterns through successive random walks in a graph: its vertices stand for the n-tuples with membership degrees at 1 and its edges link n-tuples that differ in one single dimension. The dense-enough patterns are completed with Data-Peeler-like patterns. Pairs of patterns are merged if the result is sufficiently dense. Finally, the best patterns are output, starting with those that most decrease the objective function, a greedy process that stops when the model would overfit the data according to the Minimal Description Length principle, a formalization of Occam's razor in which the best model (and its parameters) for a given set of data is the one that leads to the best compression of the data [Grünwald, 2007]. The membership degrees in an uncertain tensor must be rounded to 0/1 before applying any algorithm cited in this section. Non-negative tensor factorization directly applies to an uncertain tensor but is inappropriate: the values in the tensor reconstructed from the factors are not constrained to be at most 1. Besides, because such a decomposition weights elements (not whole patterns), it is harder to interpret than a disjunctive box cluster model.

2.3 Disjunctive Box Cluster Model

Mirkin and Kramarenko [2011] map pattern mining in 0/1 tensors to a regression problem: the set $\mathcal{X}$ of relevant patterns explains the membership degrees in the tensor T. More precisely, a parameter $\lambda_X \in \mathbb{R}$ is estimated for every pattern $X \in \mathcal{X}$ and the regression model, called *disjunctive box cluster model*, predicts that T_t is:

$$\hat{T}_t = \begin{cases} \max\limits_{X \in \mathcal{X} \text{ s.t. } t \in X} \lambda_X + \lambda_0 & \text{if } \exists X \in \mathcal{X} \text{ s.t. } t \in X \\ \lambda_0 & \text{otherwise} \end{cases} \tag{2.1}$$

The analyst fixes the intercept $\lambda_0 \in [0, 1]$ of the model. A greater λ_0 favors the discovery of a model with more, smaller, and denser patterns. The average membership degree in T minimizing $RSS_T(\emptyset)$, it should be considered the default value of λ_0. From a biclustering point of view, the uncertain matrix can be seen as a similarity matrix and

λ_0 becomes a similarity shift, i. e., a constant that is subtracted to every similarity. Model 2.2 predicts that $T_t - \lambda_0$ is a value that is independent from λ_0:

$$\hat{T}_t - \lambda_0 = \begin{cases} \max\limits_{X \in \mathcal{X} \text{ s.t. } t \in X} \lambda_X & \text{if } \exists X \in \mathcal{X} \text{ s.t. } t \in X \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

Ordinary least squares guide the selection of the model, i. e., the model aims to minimize the residual sum of squares, denoted RSS_T:

$$RSS_T(\mathcal{X}) = \sum_{t \in \prod_{i=1}^n D_i} (T_t - \hat{T}_t)^2 \tag{2.3}$$

Encoding $\mathcal{X}$ in the Boolean factors A^1, ..., A^n of a rank-$|\mathcal{X}|$ Boolean CP decomposition (see Section 2.2), $RSS_T(\mathcal{X})$ can be rewritten as: $RSS_T(A^1, \ldots, A^n) = \|T - \lambda_0 \mathbb{1} - \max_{k=1}^{|\mathcal{X}|} \lambda_k A^1_{:k} \otimes \cdots \otimes A^n_{:k}\|^2$. With λ_0 set to 0, the Boolean CP decomposition of a tensor therefore aims to minimize RSS_T with $\forall k \in \{1, \ldots, |\mathcal{X}|\}$, $\lambda_k = 1$. In contrast, those are parameters in the disjunctive box cluster model: they are estimated so that RSS_T is minimized.

The *TriclusterBox* algorithm [Mirkin and Kramarenko, 2011] repeatedly searches for one single pattern X that locally minimizes $X \mapsto RSS_T(\{X\})$. In that framework, breaking the residual sum of squares into a sum over the n-tuples $t \in X$, where $\hat{T}_t = \lambda_X + \lambda_0$, and a sum over the n-tuples $t \notin X$, where $\hat{T}_t = \lambda_0$:

$$\begin{aligned} RSS_T(\{X\}) &= \sum_{t \in X} (T_t - \lambda_X - \lambda_0)^2 + \sum_{t \in \left(\prod_{i=1}^n D_i\right) \setminus X} (T_t - \lambda_0)^2 \\ &= \sum_{t \in X} \left((T_t - \lambda_0)^2 - 2\lambda_X (T_t - \lambda_0) + \lambda_X^2 \right) + \sum_{t \in \left(\prod_{i=1}^n D_i\right) \setminus X} (T_t - \lambda_0)^2 \\ &= \sum_{t \in \prod_{i=1}^n D_i} (T_t - \lambda_0)^2 - 2\lambda_X \sum_{t \in X} (T_t - \lambda_0) + |X| \lambda_X^2 \end{aligned}$$

Given an arbitrary pattern X, λ_X must minimize $\lambda_X \mapsto RSS_T(\{X\})$:

$$\begin{aligned} \frac{\partial RSS_T(\{X\})}{\partial \lambda_X} &= 0 \\ -2 \sum_{t \in X} (T_t - \lambda_0) + 2|X|\lambda_X &= 0 \\ \lambda_X &= \frac{\sum_{t \in X} (T_t - \lambda_0)}{|X|} \end{aligned}$$

Translated into English, λ_X is simply the density of the pattern X after the similarity shift, a value that the analyst easily interprets.

Given that optimal λ_X for any pattern X, the pattern that minimizes $X \mapsto RSS_T(\{X\})$ is as well the pattern that maximizes the function $g : X \mapsto |X|\lambda_X^2$, i. e., after the similarity shift, the area times the density squared:

$$\begin{aligned}
\arg\min_X RSS_T(\{X\}) &= \arg\min_X \sum_{t \in \prod_{i=1}^n D_i} (T_t - \lambda_0)^2 - 2\lambda_X \sum_{t \in X} (T_t - \lambda_0) + |X|\lambda_X^2 \\
&= \arg\min_X -2\lambda_X \sum_{t \in X} (T_t - \lambda_0) + |X|\lambda_X^2 \\
&= \arg\min_X -2\lambda_X |X|\lambda_X + |X|\lambda_X^2 \\
&= \arg\min_X -|X|\lambda_X^2 \\
&= \arg\max_X |X|\lambda_X^2 \\
&= \arg\max g
\end{aligned}$$

TriclusterBox searches for patterns that locally maximize g. It starts with patterns that involve one single element in the $n-1$ first dimensions. In the n^{th} dimension, an initial pattern involves all elements such that it only includes n-tuples with membership degrees equal to 1. Each of those $\prod_{i=1}^{n-1} |D_i|$ patterns is the starting point of a hill-climbing procedure: every iteration leads to a pattern that involves one more or one less element, chosen so that that g increases as much as possible. Once g cannot increase, hill-climbing stops and the current pattern is added to the set $\mathcal{X}$ to return.

Chapter 3

Bigfoot

The `Bigfoot` algorithm aims to grow small patterns, e. g., computed by `multidupehack`, into more relevant patterns (patterns that are larger and denser). From now on, the small patterns at the input of `Bigfoot` (and, in the next chapter, PAF) are called *fragments*. Like *TriclusterBox* [Mirkin and Kramarenko, 2011], `Bigfoot` uses hill-climbing to repeatedly search for one pattern that locally minimizes the residual sum of squares (2.3) of the regression model (2.2), i. e., that locally maximizes $g : X \mapsto |X|\lambda_X^2$ with $\lambda_X = \frac{\sum_{t \in X}(T_t - \lambda_0)}{|X|}$. However, the *locality* is differently defined. *TriclusterBox* [Mirkin and Kramarenko, 2011] considers two patterns $X = \prod_{i=1}^{n} X_i$ and $Y = \prod_{i=1}^{n} Y_i$ neighbors (i. e., an iteration of *TriclusterBox* can go from X to Y) if and only if $|(\cup_{i=1}^{n} X_i) \Delta (\cup_{i=1}^{n} Y_i)| = 1$, where Δ denotes the symmetric difference. In contrast, in this section, X and Y are neighbors (i. e., an iteration of `Bigfoot` can go from X to Y) if and only if they both include the input pattern fragment to grow and $|\cup_{i=1}^{n} X_i| - |\cup_{i=1}^{n} Y_i| = \pm 1$. Unless X is the pattern fragment to grow, X therefore has more neighbors: `Bigfoot` more thoroughly explores the pattern space than *TriclusterBox* and usually discovers patterns with higher g values.

Table 3.1 illustrates the difference between *TriclusterBox*'s hill-climbing and `Bigfoot`'s. Suppose that the first three iterations of hill-climbing grow the (2×2)-fragment F_1 (darker cells), with $|F_1|\lambda_{F_1}^2 = 4\left(\frac{4}{4}\right)^2 = 4$, into the (3×4)-pattern P_1 (all gray cells), with $|P_1|\lambda_{P_1}^2 = 12\left(\frac{10}{12}\right)^2 \approx 8.333$. `Bigfoot`'s next iteration searches for a pattern of size 4×4 or 3×5 that contains F_1, whereas *TriclusterBox*'s next iteration searches for a pattern (of size 4×4 or 3×5 too) that contains P_1 or for a pattern (of size 2×4 or 3×3) that is contained in P_1. Given the size of the whole matrix, `Bigfoot` therefore considers 30 candidate patterns to find the one with the greatest area times density squared, whereas *TriclusterBox* only considers 11 candidate patterns, one candidate per element in $D_1 \cup D_2$. Table 3.2 depicts the pattern P_2 that

	c1	c2	c3	c4	c5	c6
r1	0	0	1	1	1	1
r2	0	0	1	0	1	1
r3	0	1	1	1	0	1
r4	1	0	1	1	1	0
r5	1	1	1	1	0	0

Table 3.1: Three iterations of hill-climbing may grow a (2×2)-fragment F_1 (darker cells) into a (3×4)-pattern P_1 (all dark cells), which is constrained to be a super-pattern of F_1.

	c1	c2	c3	c4	c5	c6
r1	0	0	1	1	1	1
r2	0	0	1	0	1	1
r3	0	1	1	1	0	1
r4	1	0	1	1	1	0
r5	1	1	1	1	0	0

Table 3.2: (4×4)-pattern P_2 (all grayed cells) with the highest area times squared density among the 30 super-patterns of F_1 (darker cells) of size 4×4 or 3×5. An iteration of `Bigfoot`'s hill-climbing goes from P_1 in Table 3.1 to P_2, whereas *TriclusterBox*'s hill-climbing terminates at P_1.

`Bigfoot` finds. Among the 30 candidate super-patterns of F_1, P_2 maximizes the objective function: $|P_2|\lambda_{P_2}^2 = 16\left(\frac{13}{16}\right)^2 \approx 10.563$. In contrast, none of the 11 candidate patterns, that *TriclusterBox* considers, has an area times density squared that exceeds that of P_1, i. e., 8.333. *TriclusterBox*'s hill-climbing therefore stops and returns P_1. That example shows that `Bigfoot` more thoroughly explores the pattern space, what allows the discover of higher local maximums of g. On the negative side, `Bigfoot`'s hill-climbing iterations take more time than *TriclusterBox*'s.

Section 3.1 presents the `Bigfoot` algorithm, its pseudo-code, and the function it maximizes. Section 3.2 defines the ILP problems solved during the hill-climbing. Finally, Section 3.3 presents `Bigfoot-lr`, which solves linear relaxation of `Bigfoot`'s ILP problems.

3.1 Algorithm

Algorithm 1 gives a big picture of `Bigfoot`. Every iteration of `Bigfoot` starts with the selection of a fragment that no previously discovered pattern, in $\mathcal{X}$, includes (line 18). Line 4 chooses that fragment as the one that, added to $\mathcal{X}$, minimizes RSS_T. It is indeed

the most promising fragment to grow. At the first iteration, it actually is the fragment with the greatest g. In the subsequent iterations, fragments that largely overlap with the previously discovered patterns are usually deemed less promising. Section 5.2 is dedicated to the efficient computation of $\arg\min_{G\in\mathcal{F}} RSS_T(\mathcal{X}\cup\{G\})$.

Algorithm 1: `Bigfoot`

Input: uncertain tensor T, set of fragments $\mathcal{F}$
Output: set of patterns $\mathcal{X}$ fitting T through model (2.2)

```
1  X ← ∅
2  F_∪ ← ∪_{F∈F} F
3  while F ≠ ∅ do
4      F ← arg min_{G∈F} RSS_T(X ∪ {G})
5      X^max ← F
6      intermediary ← true
7      while intermediary do
8          intermediary ← false
9          (S_1, ..., S_n) ← (|X_1^max|, ..., |X_n^max|)
10         for i = 1 → n do
11             S_i ← S_i + 1
12             X ← f(T, F_∪, F, g(X^max), S_1, ..., S_n)
13             if g(X) > g(X^max) then
14                 X^max ← X
15                 intermediary ← true
16             S_i ← S_i − 1
17     X ← X ∪ {X^max}
18     F ← {F ∈ F | F ⊈ X^max}
19 return X
```

Once the fragment F chosen, `Bigfoot` searches for a pattern $X^{\max} \supseteq F$ that locally maximizes g, in the sense specified at the beginning of this chapter. Starting with F (line 5), lines 5–16 implement a hill-climbing maximization of g in the space of the number of elements $(S_1, \ldots, S_n) \in \mathbb{N}^n$ that the super-patterns of F involve, in each of the n dimensions. Any of those numbers (line 10) can be incremented (line 11) during the search. Among super-patterns $X = \prod_{i=1}^{n} X_i$ of F with $|X_1|$, ..., $|X_n|$ matching the last n arguments of the function f (line 12), called n times per iteration of the hill-climbing, f returns the one maximizing g.

Figure 3.1 illustrates the hill-climbing maximization of the non-linear function $g : X \mapsto |X|\lambda_X^2$ when `Bigfoot` grows the fragment F_2, in Table 1.2. That fragment, which involves 3 regions and 2 chocolate types, must first grow into a pattern of sizes either 4×2 (one more region) or 3×3 (one more chocolate type). In that case,

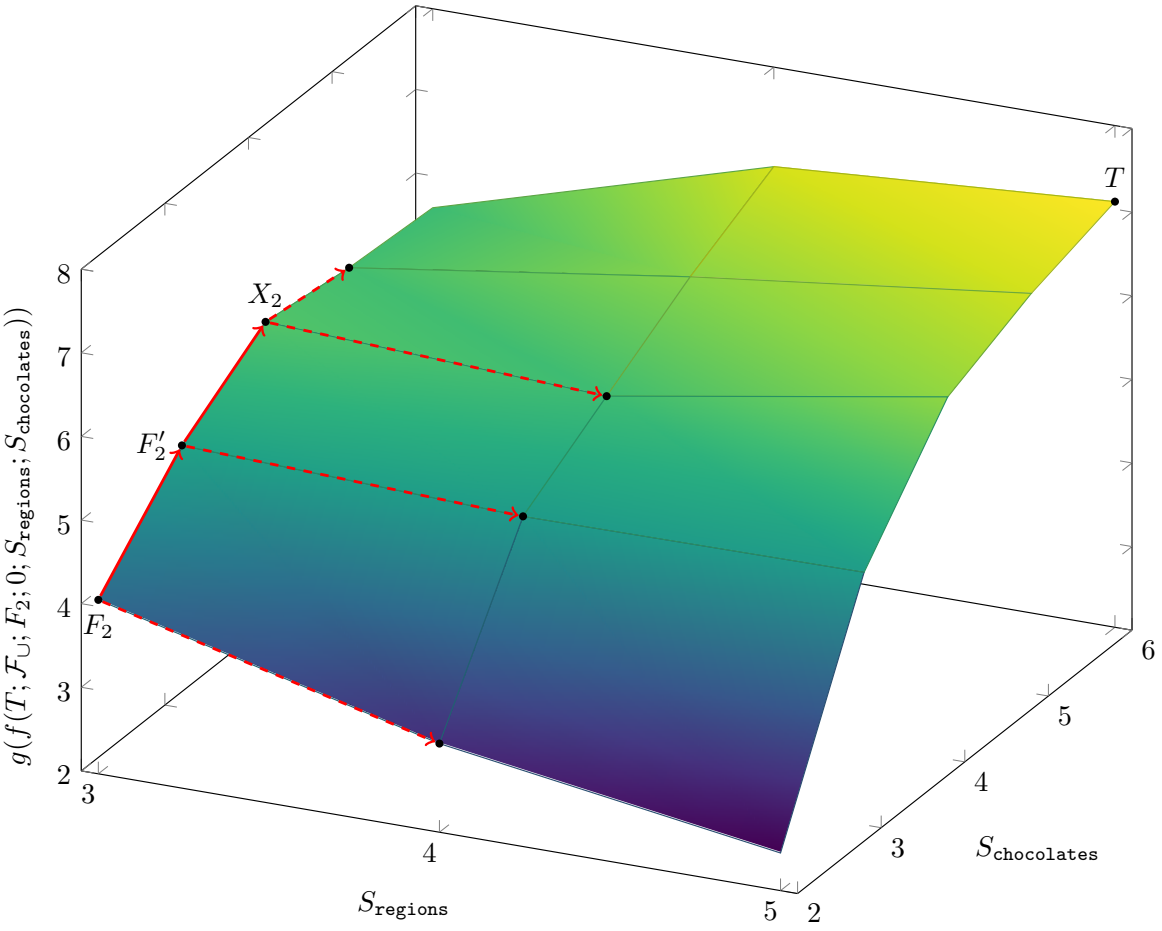


Figure 3.1: Hill-climbing maximization of $g : X \mapsto |X|\lambda_X^2$ in the space of the number of elements (S_1, S_2).

the super-pattern of F_2 that maximizes g is the one (denoted F_2' in the figure) that involves one more chocolate type: `sweet`. The subsequent iteration of the hill-climbing procedure is, here, the last one. It results in the discovery of the pattern X_2, which locally maximizes g. Notice that the hill-climbing procedure terminates before the discovery of g's global maximum, which is, in this example, obtained for the pattern that includes all the 2-tuples of the uncertain matrix. That illustrates two potential problems:

1. Hill-climbing may prematurely terminate: given a fragment F to grow, this optimization technique does not necessarily return the super-pattern of F that globally maximizes g. Only a *local* maximum of g is guaranteed to be discovered.

2. When $n = 2$, i. e., for patterns in uncertain *matrices*, maximizing $g : X \mapsto |X|\lambda_X^2$ apparently favors too large patterns, which go over the edges of the "desired" patterns. Consider, for instance, two patterns X and Y that have the same

density, involve the same number of elements in each of the n dimensions and do not share any element. Then, $|X \sqcup Y| = 2^n|X|$ and $\lambda_{|X \sqcup Y|} \geq \frac{2|X|\lambda_X}{2^n|X|} = \frac{\lambda_X}{2^{n-1}}$. As a consequence, $g(X \sqcup Y) \geq \frac{g(X)}{2^{n-2}}$. If $n = 2$, $g(X \sqcup Y) \geq g(X)$, i. e., $X \sqcup Y$ is preferred over X (and Y), whatever the values T_t for $t \in (X \sqcup Y) \setminus (X \cup Y)$, which can possibly be all null. Since $g(X \sqcup Y)$ exponentially decreases when n increases, this problem is quite unusual for $n \geq 3$.

Assuming that every pattern of interest contains at least one fragment that no other pattern contains and there is no fragments of non-interesting patterns, the both problems are hopefully solved. Growing each fragment gives all interesting patterns except the (too) large ones, since there is no fragment that leads to the local (or optimal) maximum of g related to them. In case of chocolates sales tensor, there is no fragment that leads to the top-right corner in figure 3.1.

3.2 ILP Model

Here is the Integer Linear Programming (ILP) problem that f solves:

$$\text{Maximize} \sum_{t \in \mathcal{F}_\cup} w_t(T_t - \lambda_0) \tag{3.1}$$

under the following constraints:

$$\forall t \in \mathcal{F}_\cup,\ 0 \leq \sum_{i=1}^{n} x_{t_i} - n w_t \leq n - 1 \tag{3.2}$$

$$\forall t \in F, \forall i \in \{1, \ldots, n\},\ x_{t_i} = 1 \tag{3.3}$$

$$\forall i \in \{1, \ldots, n\},\ \sum_{e \in \{t_i | t \in \mathcal{F}_\cup\}} x_e = S_i \tag{3.4}$$

$$\forall t \in \mathcal{F}_\cup,\ w_t \in \{0, 1\} \tag{3.5}$$

$$\forall e \in \cup_{i=1}^{n} \{t_i \mid t \in \mathcal{F}_\cup\},\ x_e \in \{0, 1\} \tag{3.6}$$

Constraints 3.5 and 3.6 force all variables w_t and x_e to be either 0 or 1. $w_t = 1$ indicates that t is in the returned pattern. $x_e = 1$ indicates that at least one of those n-tuples, with $w_t = 1$, involves the element e. f therefore returns the pattern $X = \prod_{i=1}^{n} \{e \in D_i \mid x_e = 1\}$. As detailed later, Constraint 3.2 forces the values of all variables w_t and x_e to be coherent with the syntactic definition of a pattern (see Section 1.2). Constraint 3.3 forces the returned pattern X to be a super-pattern

of the grown fragment F, i. e., $X \supseteq F$. Constraint 3.4 forces the returned pattern $X = \prod_{i=1}^{n} X_i$ to have $\forall i \in \{1, \ldots, n\}, |X_i| = S_i$. Given those constant cardinalities, $|X|$ is constant too. It is $\prod_{i=1}^{n} S_i$. That is why maximizing $g(X) = |X|\lambda_X^2 = \frac{(\sum_{t \in X}(T_t - \lambda_0))^2}{|X|}$ amounts to maximizing $\sum_{t \in X}(T_t - \lambda_0)$.

Note that f maximizes $\sum_{t \in \mathcal{F}_\cup} w_t(T_t - \lambda_0)$, which is slightly different from $\sum_{t \in X}(T_t - \lambda_0)$: n-tuples that are not in $\mathcal{F}_\cup$ (i. e., not found in any fragment) are missing. f therefore assumes that $\forall t \notin \mathcal{F}_\cup$, $T_t = \lambda_0$. Hopefully, the fragments include all the n-tuples in the desired patterns, i. e., $\cup_{X \in \mathcal{X}} X \subseteq \mathcal{F}_\cup$. If so, the assumption is perfectly coherent with Model (2.2), which predicts $\forall t \notin \cup_{X \in \mathcal{X}} X$, $\hat{T}_t = \lambda_0$. However the fragments do not *need* to include all the n-tuples in the patterns of interest. In particular, f returns $\prod_{i=1}^{n} \{e \in D_i \mid x_e = 1\}$ (and *not* $\{t \in \mathcal{F}_\cup \mid w_t = 1\}$), which *can* include n-tuples that are not in $\mathcal{F}_\cup$.

Even if the assumption "$T_t = \lambda_0$ if $t \notin \mathcal{F}_\cup$" is coherent with Model (2.2), its mere presence requires a justification. It is simple: for f's computation to be fast, the number of variables in the ILP problem ought to be small. Thanks to the assumption, there are "only" $|\mathcal{F}_\cup|$ variables w_t (Constraint 3.5) and $|\{t_i \mid t \in \mathcal{F}_\cup\}|$ variables x_e (Constraint 3.6) rather than, respectively, $\prod_{i=1}^{n} |D_i|$ and $\sum_{i=1}^{n} |D_i|$ without the assumption. A smaller number of x_e variables is particularly interesting. Indeed, Constraint 3.2 is equivalent to:

$$\forall t \in \mathcal{F}_\cup,\ w_t = 1 \Leftrightarrow \forall i \in \{1, \ldots, n\},\ x_{t_i} = 1 \ .$$

A valuation of the x_e variables therefore implies a unique valuation of the w_t variables, and reciprocally. Thanks to that, the ILP solver only needs to branch on the x_e variables, which usually are far less numerous: $|\cup_{i=1}^{n} \{t_i \mid t \in \mathcal{F}_\cup\}|$ of x_e variables vs. $|\mathcal{F}_\cup|$ of w_t variables.

Although $g(X^{\max})$ is an argument of f, it does not appear in the ILP model above. One constraint is indeed added to fasten the resolution of the ILP problem without altering Algorithm 1's output:

$$\sum_{t \in \mathcal{T}_\cup} w_t(T_t - \lambda_0) > \sqrt{g(X^{\max}) \prod_{i=1}^{n} S_i} \tag{3.7}$$

$\prod_{i=1}^{n} S_i$ is the area of the pattern X that f will return, i. e., $|X|$. Under the assumption "$T_t = \lambda_0$ if $t \notin \mathcal{F}_\cup$", Constraint 3.7 therefore is equivalent to $g(X) > g(X^{\max})$. Since f's output only matters if the test at line 13 of Algorithm 1 passes, Constraint 3.7 safely prunes the search space. It enforces a lower-bound on the values of g that are acceptable, what prevents the ILP solver (which uses a branch-and-cut algorithm) from

considering patterns that are worse than the best pattern found so far, at the previous iteration of the hill-climbing procedure.

3.3 Bigfoot-lr

To solve the ILP problem above, f may take exponential time. For a polynomial time complexity, that problem can be linearly relaxed, i. e., Constraint 3.5 is substituted by:

$$\forall t \in \mathcal{F}_\cup,\, w_t \in \mathbb{R} \ .$$

To maximize $\sum_{t\in\mathcal{F}_\cup} w_t T_t$ (Problem 3.1) while satisfying the first inequality of Constraint 3.2 (the only constraint involving the w_t variables), w_t is easily computed for all $t \in \mathcal{F}_\cup$. It simply is $\frac{\sum_{i=1}^{n} x_{t_i}}{n}$. The problem can therefore be rewritten as the maximization of:

$$\sum_{t\in\mathcal{F}_\cup} \frac{\sum_{i=1}^{n} x_{t_i}}{n}(T_t - \lambda_0) = \frac{1}{n}\sum_{i=1}^{n}\sum_{t\in\mathcal{F}_\cup} x_{t_i}(T_t - \lambda_0) \ .$$

The n-tuples in $\mathcal{F}_\cup$ can be taken slice per slice (definition in Section 1.2), i. e., the linearly relaxed problem is:

$$\text{Maximize } \frac{1}{n}\sum_{i=1}^{n}\sum_{e\in D_i}\left(x_e \sum_{t\in\mathcal{F}_\cup \text{ s.t. } t_i=e}(T_t - \lambda_0)\right) \tag{3.8}$$

The innermost sums are constants. They only need to be computed once, at the beginning of Algorithm 1. Solving Problem 3.8 under the (remaining) constraints 3.3, 3.4 and 3.6 is trivial. Constraint 3.3 forces $x_e = 1$ for all elements e involved in the grown fragment and additional x_e variables are set to 1 until Constraint 3.3 is satisfied: the variables x_e with e in the proper dimension and the greatest $\sum_{t\in\mathcal{F}_\cup \text{ s.t. } t_i=e}(T_t - \lambda_0)$. Algorithm 2 gives the pseudo-code of that solution. It runs in $O(\sum_{i=1}^{n} S_i)$ time.

Algorithm 2: f with the linear relaxation

Input: $D_1, \ldots, D_n$ ordered w.r.t. $\sum_{t\in\mathcal{F}_\cup \text{ s.t. } t_i=e}(T_t - \lambda_0)$, fragment F, numbers of elements $(S_1, \ldots, S_n)$

Output: Pattern X, solution to the constrained Prob. 3.8

```
for i = 1 → n do
    X_i ← {t_i | t ∈ F}
    X_i ← X_i ∪ {last S_i − |X_i| elements in D_i \ X_i}
return ∏_{i=1}^{n} X_i
```

`Bigfoot-lr` is the name given to Algorithm 1 when f greedily solves the linearly relaxed problem. Only growing the fragments with n-tuples in the densest slices of $\mathcal{F}_\cup$ is naive. Section 6 shows that `Bigfoot-lr` poorly performs: its hill-climbing searches terminate after a few iterations and it returns patterns that are only slightly larger fragments of the interesting patterns. However `Bigfoot` can achieve the work, i. e., it can further grow the patterns that `Bigfoot-lr` outputs. Being larger than the initial fragments, `Bigfoot-lr`'s patterns save some iterations of `Bigfoot`'s hill-climbing. On the other hand, `Bigfoot-lr`'s patterns, taken altogether, include more n-tuples than the initial fragments. As a consequence, solving the individual ILP problems takes more time. *TriclusterBox* [Mirkin and Kramarenko, 2011], discussed in Section 2.2, is less naive than `Bigfoot-lr`. However, it cannot provide "larger fragments" to `Bigfoot`. Indeed, those fragments would not grow because they locally maximize g.

Chapter 4

PAF

Papas At Farm or simply PAF hierarchically agglomerates the fragments it is given to construct candidate explanatory variables for the disjunctive box cluster model. Those candidate variables are the patterns at the nodes of the built dendrogram (see Figure 1.2). As for `Bigfoot`, a complete algorithm can provide the initial set of fragments, the leaves of the dendrogram. Being a clustering algorithm, PAF relies on a similarity measure, here a similarity between patterns. It agglomerates, iteration after iteration, the two most similar patterns, what results in the smallest pattern containing both. Section 4.1 defines and justifies the designed similarity measure that PAF uses. Section 4.2 gives PAF's pseudo-code and illustrates its execution on a small set of fragments. Finally, Section 4.3 presents a pre-processing step, which drastically speeds up the subsequent execution of PAF and allows to sum up large uncertain tensors, in which many fragments were mined.

4.1 Similarity Between Patterns

Besides the agglomeration operator $\sqcup$, defined in Section 1.2, a hierarchical agglomeration only requires the definition of a similarity. Here, the hierarchical agglomeration aims to generate patterns that are good candidate variables for the final disjunctive box cluster model (2.2). To measure the similarity between two patterns X and Y, it therefore makes sense to contrast the residual sum of squares (2.3) of a model involving X and Y with that of the model where $X \sqcup Y$ substitutes X and Y. The distance[1]

[1]The term *distance* is here abused. d_T only is a similarity measure, smaller values meaning more similar.

$d_T(X, Y)$ between X and Y is here defined as

$$d_T(X, Y) = RSS_T(\{X \sqcup Y\}) - RSS_T(\{X, Y, X \sqcup Y\}). \tag{4.1}$$

Both models, $\{X \sqcup Y\}$ and $\{X, Y, X \sqcup Y\}$, only depend on X and Y. Indeed, although all pattern fragments to agglomerate are known and some agglomerates may have already been generated, they will not necessarily be part of the final model. Like in *TriclusterBox* [Mirkin and Kramarenko, 2011] and in `Bigfoot`, $\lambda_P = \frac{\sum_{t \in P}(T_t - \lambda_0)}{|P|}$, for all pattern P in the tensor T. As explained earlier, that value, which minimizes $RSS_T(\{P\})$, is easy to interpret: it is the density of P after the similarity shift by λ_0.

Contrasting the residual sum of squares of the models $\{X \sqcup Y\}$ and $\{X, Y\}$ (instead of $\{X, Y, X \sqcup Y\}$) may look more natural. Nevertheless, it would amount to adding $\sum_{t \in (X \sqcup Y) \setminus (X \cup Y)} \left((T_t - \lambda_{X \sqcup Y})^2 - (T_t - \lambda_0)^2\right)$ to $d_T(X, Y)$ and the similarity between X and Y would increase with $|(X \sqcup Y) \setminus (X \cup Y)|$. As a consequence, PAF would favor the agglomeration of patterns with smaller intersections to directly construct large agglomerates, possibly missing intermediary denser patterns that could be more relevant for the final model.

If λ_X and λ_Y are both greater than $\lambda_{X \sqcup Y}$ (agglomerating two patterns usually gives a sparser pattern), then, using the inclusion-exclusion principle (to rewrite $d_T(X, Y)$ with three sums over X, Y and $X \cap Y$) and the equality $\sum_{t \in P}(T_t - \lambda_0) = |P|\lambda_P$, Equation (4.1) becomes:

$$\begin{aligned} & |X|(\lambda_X - \lambda_{X \sqcup Y})^2 + |Y|(\lambda_Y - \lambda_{X \sqcup Y})^2 \\ + \quad & |X \cap Y|(\min(\lambda_X, \lambda_Y)^2 - \lambda_{X \sqcup Y}^2) \\ - \quad & 2(\min(\lambda_X, \lambda_Y) - \lambda_{X \sqcup Y}) \textstyle\sum_{t \in X \cap Y}(T_t - \lambda_0). \end{aligned}$$

If only λ_X (resp. λ_Y) is greater than $\lambda_{X \sqcup Y}$, $d_T(X, Y)$ is $|X|(\lambda_{X \sqcup Y} - \lambda_X)^2$ (resp. $|Y|(\lambda_{X \sqcup Y} - \lambda_Y)^2$). If both are lesser than $\lambda_{X \sqcup Y}$, $d_T(X, Y) = 0$. The following matrix gives the distances between the fragments in Table 1.3:

$$d_T(\mathcal{F}, \mathcal{F}) = \begin{array}{c} \\ F_1 \\ F_2 \\ F_3 \\ F_4 \\ F_5 \end{array} \begin{array}{c} \begin{array}{ccccc} F_2 & F_3 & F_4 & F_5 & F_6 \end{array} \\ \begin{pmatrix} 0.003 & 0.169 & 10^{-5} & 0.010 & 0.115 \\ & 0.107 & 0.003 & 0.007 & 0.093 \\ & & 0.125 & 0.086 & 0.005 \\ & & & 0.011 & 0.100 \\ & & & & 0.059 \end{pmatrix} \end{array} \tag{4.2}$$

Each value is the distance between the fragment in row and the fragment in column. Here, the two closest fragments are F_1 and F_4, with $d_t(F_1, F_4) = 10^{-5}$.

4.2 Hierarchical Agglomeration

Algorithm 3 describes PAF's hierarchical agglomeration of the input pattern fragments. Lines 2–3 insert in a set $\mathcal{C}$ of candidates for agglomeration every subset of two input fragments. Line 5 takes in $\mathcal{C}$ the patterns X and Y that minimize d_T. $X \sqcup Y$ is output (line 6) because it enters the dendrogram. More precisely, $X \sqcup Y$ substitutes all its sub-patterns in the frontier $\mathcal{F}$ of the dendrogram (line 7 and 11). Every pair of candidates that involves at least one of the patterns that $X \sqcup Y$ substituted is not a candidate pair anymore (line 8). On the contrary, $X \sqcup Y$ and every other pattern in $\mathcal{F}$ form a new candidate pair (lines 9-10). A new iteration starts (line 5 takes in $\mathcal{C}$ the two patterns that minimize d_T, etc.), unless the hierarchical agglomeration is over ($\mathcal{C} = \emptyset$ at line 4).

Algorithm 3: PAF

Input: uncertain tensor T, set of fragments $\mathcal{F}$
Output: patterns (variables for Model (2.2))

```
1  C ← ∅                    /* a self-balancing binary search tree */
2  forall {X, Y} ⊆ F do
3      insert {X, Y} in C with key d_T(X, Y)
4  while C ≠ ∅ do
5      {X, Y} ← first element in C
6      output X ⊔ Y
7      F ← {F ∈ F | F ⊈ X ⊔ Y}
8      C ← {{A, B} ∈ C | {A, B} ⊆ F}
9      forall F ∈ F do
10         insert {F, X ⊔ Y} in C with key d_T(F, X ⊔ Y)
11     F ← F ∪ {X ⊔ Y}
```

To not have to search in $\mathcal{C}$ the candidate pair that minimizes d_T, $\mathcal{C}$ is maintained ordered by increasing d_T. To efficiently compute those distances (formulas at the end of Section 4.1), the elements in a dimension are arbitrarily ordered (for linear time intersections/unions), $T - \lambda_0 \mathbb{1}$ is stored in a trie, any pattern $X = \prod_{i=1}^{n} X_i$ in $\mathcal{F}$ is stored as $(X_1, \ldots, X_n)$ along its area $|X|$ and $\sum_{t \in X}(T_t - \lambda_0)$, and so is a candidate agglomerate. Finally, every pattern in $\mathcal{F}$ is linked to the pairs in $\mathcal{C}$ that include it so that line 8 is efficiently computed.

In the running example, PAF starts with the agglomeration of F_1 and F_4, which are the two closest fragments. The resulting pattern, $F_1 \sqcup F_4 =$

$\{\texttt{Southest}, \texttt{West} - \texttt{Center}\} \times \{\texttt{bitter}, \texttt{crunchy}, \texttt{semisweet}\}$, substitutes F_1 and F_4 in the frontier of the built dendrogram. The other patterns, F_2, F_3, F_5 and F_5, not being included in $F_1 \sqcup F_4$, they stay in the frontier and their distances to $F_1 \sqcup F_4$ are computed. At the beginning of the second iteration of the hierarchical agglomeration, the distances between the patterns in the frontier are:

$$\begin{array}{c} \\ F_2 \\ F_3 \\ F_5 \\ F_6 \end{array} \begin{array}{c} \begin{array}{cccc} F_3 & F_5 & F_6 & F_1 \sqcup F_4 \end{array} \\ \begin{pmatrix} 0.107 & 0.007 & 0.093 & 0.003 \\ & 0.086 & 0.005 & 0.015 \\ & & 0.059 & 0.010 \\ & & & 0.107 \end{pmatrix} \end{array} \qquad \text{(Iteration 2)}$$

F_2 and $F_1 \sqcup F_4$ being the two closest patterns, their agglomeration, $F_1 \sqcup F_4 \sqcup F_2$, enters the frontier of the dendrogram. It substitutes F_2 and $F_1 \sqcup F_4$. The considered distances at the beginning of the third iteration are:

$$\begin{array}{c} \\ F_3 \\ F_5 \\ F_6 \end{array} \begin{array}{c} \begin{array}{ccc} F_5 & F_6 & F_1 \sqcup F_4 \sqcup F_2 \end{array} \\ \begin{pmatrix} 0.086 & 0.005 & 0.120 \\ & 0.059 & 0.008 \\ & & 0.010 \end{pmatrix} \end{array} \qquad \text{(Iteration 3)}$$

This time, F_3 and F_6 are the two closest patterns. After the related update of the dendrogram frontier, the distances between its patterns are:

$$\begin{array}{c} \\ F_5 \\ F_1 \sqcup F_4 \sqcup F_2 \end{array} \begin{array}{c} \begin{array}{cc} F_1 \sqcup F_4 \sqcup F_2 & F_3 \sqcup F_6 \end{array} \\ \begin{pmatrix} 0.008 & 0.062 \\ & 0.099 \end{pmatrix} \end{array} \qquad \text{(Iteration 4)}$$

The process goes on with the agglomeration of F_5 and $F_1 \sqcup F_4 \sqcup F_2$ and, finally, with the agglomeration of $F_1 \sqcup F_4 \sqcup F_2 \sqcup F_5$ and $F_3 \sqcup F_6$, which are 0.041-distant from each other. In that example, the agglomeration of all input fragments, the root of the dendrogram, contains all the tuples in the uncertain matrix. It is not the case when some element is not be involved in any fragment. Figure 1.2, at the end of Chapter 1, depicts the built dendrogram. The nodes of the dendrogram are then candidates to explanatory variables for Mirkin and Kramarenko's regression model regardless of their hierarchical structure.

4.3 Growing the Pattern Fragments before the Agglomeration

PAF computes a number of distances that depends on the square of the number of pattern fragments it is given. PAF's memory consumption, dominated by the storage of $\mathcal{C}$, depends on $|\mathcal{F}|^2$ too. Those computational costs may be prohibitive. To have a constant $m \in \mathbb{N}$ upper bound $|\mathcal{C}|$, Algorithm 3 must be given at most $\sqrt{m}$ patterns to agglomerate. Algorithm 4 does so. It takes the fragments at input, grows them, and returns at most $\sqrt{m}$ patterns.

Algorithm 4: PAF's pre-processing

Input: uncertain tensor T, fragments $\mathcal{F}$, $m \in \mathbb{N}$
Output: patterns (variables for Model (2.2))

1 $\mathcal{C} \leftarrow \emptyset$ `/* a self-balancing binary search tree */`
2 $\mathcal{H} \leftarrow \emptyset$ `/* a hash set */`
3 **forall** $X = \prod_{i=1}^{n} X_i \in \mathcal{F}$ **do**
4 **output** X
5 $e \leftarrow \arg\max_{f \in \cup_{i=1}^{n} D_i \setminus X_i} \lambda_{X^{\cup\{f\}}}$
6 insert (X, e) in $\mathcal{C}$ with key $d_T(X, X^{\cup\{e\}})$
7 **while** $|\mathcal{C}| > \sqrt{m}$ **do**
8 $(X, e) \leftarrow \mathcal{C}$'s first entry taken out of $\mathcal{C}$
9 $X \leftarrow X^{\cup\{e\}}$
10 **output** $X = \prod_{i=1}^{n} X_i$
11 $e \leftarrow \arg\max_{f \in \cup_{i=1}^{n} D_i \setminus X_i} \lambda_{X^{\cup\{f\}}}$
12 **if** $X^{\cup\{e\}} \notin \mathcal{H}$ **then**
13 $\mathcal{H} \leftarrow \mathcal{H} \cup \{X^{\cup\{e\}}\}$
14 insert (X, e) in $\mathcal{C}$ with key $d_T(X, X^{\cup\{e\}})$
15 **return** $\{X \mid (X, e) \in \mathcal{C}\}$

It greedily adds one element to one of the n dimensions of a pattern $X = \prod_{i=1}^{n} X_i$. If that element e is taken in the j^{th} dimension (i. e., $e \in D_j \setminus X_j$), then $X^{\cup\{e\}} = (\prod_{i=1}^{j-1} X_i) \times (X_j \cup \{e\}) \times (\prod_{i=j+1}^{n} X_i)$ is obtained. Using the rationale in Section 4.1, the best element e to add to a dimension of X minimizes $d_T(X, X^{\cup\{e\}})$. Here, $X \sqcup X^{\cup\{e\}} = X^{\cup\{e\}}$ and $d_T(X, X^{\cup\{e\}}) = RSS_T(\{X^{\cup\{e\}}\}) - RSS_T(\{X, X^{\cup\{e\}}\})$. $X^{\cup\{e\}}$ is usually sparser than X, i. e., $\lambda_{X^{\cup\{e\}}} < \lambda_X$. If so, $d_T(X, X^{\cup\{e\}}) = |X|(\lambda_{X^{\cup\{e\}}} - \lambda_X)^2$. If not, $d_T(X, X^{\cup\{e\}}) = 0$. In both cases, the element e that maximizes $\lambda_{X^{\cup\{e\}}}$ minimizes $d_T(X, X^{\cup\{e\}})$.

That is why lines 5 and 11 associate to a pattern X the element $e = \arg\max_{f \in \cup_{i=1}^{n} D_i \setminus X_i} \lambda_{X^{\cup\{f\}}}$. At line 5, X is an input fragment and (X, e) is inserted

in a set $\mathcal{C}$ of candidates to grow (line 6). At line 11, X is a pattern that has just grown (line 9). It was chosen for minimizing the distance (line 8). Line 14 does not insert in $\mathcal{C}$ every candidate constructed from a recently grown pattern. Indeed, line 12 imposes a condition: no previous iteration must have inserted (line 13) that same candidate. Since complete algorithms provide many overlapping fragments of every large pattern to ultimately discover in the uncertain tensor T, many of the input fragments quickly grow into a same pattern and the number of candidates decreases. Once there are less than $\sqrt{m}$ candidates (line 7), Algorithm 4 terminates. Those candidates are not grown (line 15). They are returned: PAF will hierarchically agglomerate them. Lines 4 and 10 output all the patterns that are candidates to grow. They are indeed explanatory variables that can be selected in the final disjunctive box cluster model (2.2).

The time the growing pre-process takes to get at most $\sqrt{m}$ larger fragments depends on how much the input fragments overlap. The minimal number of iterations to grow two fragments $X = \prod_{i=1}^{n} X_i$ and $Y = \prod_{i=1}^{n} Y_i$ into a same larger fragment corresponds to the case where this larger fragment is $X \sqcup Y$. The related number of iterations is $\sum_{i=1}^{n} |X_i \setminus Y_i| + |Y_i \setminus X_i| = \sum_{i=1}^{n} |X_i \Delta Y_i|$, where Δ denotes the symmetric difference. For an efficient pre-process, only fragments of a single large pattern (which may enter the final disjunctive box cluster model) should grow into a same larger fragment. For that, m must be larger than the (unknown) number of large patterns. On the other hand, the motivation behind the pre-process is to reduce the number of fragments that PAF hierarchically agglomerates. In particular, the (known) available memory must be enough to store the m candidate patterns that line 3 of Algorithm 3 generates. For that, m must be small enough. Since the available memory is known and the number of large patterns is not, m is, in practice, fixed to a large value that still allows PAF to not require more than the available memory.

Chapter 5

Forward Selection

5.1 The Forward Selection Algorithm

The disjunctive box cluster model composed of all the patterns that `Bigfoot`, `Bigfoot-lr`, `Bigfoot-lr`+`Bigfoot` or PAF (and its pre-process) outputs is hard to interpret (too much information) and it overfits the uncertain tensor. A well-chosen subset of those patterns often makes a better model of the data. Forward selection, a stepwise regression technique, is a solution. Instead of minimizing the residual sum of squares (2.3), it aims to minimize a measure that penalizes the use of too many explanatory variables, i. e., of too many patterns in Model (2.2). The *Akaike information criterion* (*AIC*) [Akaike, 1974] is such a measure. Assuming that, for any subset $\mathcal{Y}$ of the output patterns, the residuals $T_t - \hat{T}_t$ follow an independent identical normal distribution with zero mean, $AIC_T(\mathcal{Y})$ is:

$$AIC_T(\mathcal{Y}) = 2|\mathcal{Y}| + |\cup_{X\in\mathcal{X}} X| \ln(RSS_{\{t\mapsto T_t | t\in\cup_{X\in\mathcal{X}} X\}}(\mathcal{Y})) \ .$$

The first term penalizes a too complex regression model: it grows with the number of patterns (the explanatory variables) in the model. The second term favors more accurate predictions of the membership degrees: it decreases with the residual sum of squares of the model. That residual sum of squares is here restricted to the n-tuples $t \in \cup_{X\in\mathcal{X}} X$ for which at least one model, among those that the forward selection can select (the subsets of $\mathcal{X}$), can predict $\hat{T}_t \neq \lambda_0$. That number of n-tuples weights the logarithm of residual sum of square: the more membership degrees to predict, the more complex the model is allowed to be. More precisely, a model with one additional pattern is preferred if that additional pattern decreases the second term by at least 2. Another reason for AIC_T to only consider the n-tuples in $\cup_{X\in\mathcal{X}} X$ is that $\cup_{X\in\mathcal{X}} X$ is

often only a small part of all the n-tuples, $\prod_{i=1}^{n} D_i$. In that situation, the assumption behind the formula above would be clearly violated unless λ_0 is chosen as the average membership degree of the n-tuples in $\prod_{i=1}^{n} D_i \setminus \cup_{X \in \mathcal{X}} X$. Those same arguments stand when it comes to assess the quality of the selected model $\mathcal{Y} \subseteq \mathcal{X}$, i. e., AIC_T should disregard an n-tuple $t \notin \cup_{Y \in \mathcal{Y}} Y$. That is why the stepwise regression ought to be recursive, executed until $\mathcal{Y} = \mathcal{X}$, a fixed point.

Algorithm 5 formalizes the recursive forward selection (the chosen stepwise regression technique) of the disjunctive box cluster model $\mathcal{Y} \subseteq \mathcal{X}$. At every iteration, line 3 in selects the pattern in $\mathcal{X}$ that, added to $\mathcal{Y}$, minimizes RSS_T (and AIC_T), i. e., that best complements the current model. At the first iteration, it is the pattern with the greatest g. In the subsequent iterations, patterns that largely overlap with the previously discovered patterns are usually deemed less interesting. Indeed, according to Model (2.2), the prediction for the membership degree of an n-tuple in previously selected patterns can only marginally improve if they all are less dense than the newly considered pattern. If adding to $\mathcal{Y}$ any more pattern, taken in $\mathcal{X}$, would increase AIC_T (line 4), the selected patterns become the candidate patterns of a new forward selection (line 5). The recursion terminates when all candidate patterns are selected (line 2). Line 7 returns them.

Algorithm 5: Forward selection

Input: uncertain tensor T, set of candidate patterns $\mathcal{X}$
Output: set of patterns $\mathcal{Y} \subseteq \mathcal{X}$ fitting T through Model (2.2)

```
1 Y ← ∅
2 while Y ≠ X do
3 |   Y ← arg min_{X∈X} RSS_T(Y ∪ {X})
4 |   if AIC_T(Y ∪ {Y}) > AIC_T(Y) then
5 |   |   return forward-select(T, Y)
6 |   Y ← Y ∪ {Y}
7 return Y
```

5.2 Computation of $\arg\min_{X \in \mathcal{X}} RSS_T(\mathcal{Y} \cup \{X\})$

Given two sets of patterns $\mathcal{X}$ and $\mathcal{Y}$, line 4 in Algorithm 1 and line 3 in Algorithm 5 select the pattern $\arg\min_{X \in \mathcal{X}} RSS_T(\mathcal{Y} \cup \{X\})$, i. e., the pattern in $\mathcal{X}$ that best complements the current model $\mathcal{Y}$. Naively computing $\arg\min_{X \in \mathcal{X}} RSS_T(\mathcal{Y} \cup \{X\})$ has a $O(|\mathcal{X}| \prod_{i=1}^{n} |D_i|)$ time complexity, to be multiplied by the number of iterations of Algorithms 1 and 5. That cost can be significantly lowered.

At a given iteration, $RSS_T(\mathcal{Y})$ is a constant. Consequently, computing $\arg\min_{X\in\mathcal{X}} RSS_T(\mathcal{Y}\cup\{X\})$ is equivalent to computing $\arg\min_{X\in\mathcal{X}}(RSS_T(\mathcal{Y}\cup\{X\}) - RSS_T(\mathcal{Y}))$. For any pattern $X \in \mathcal{X}$, $RSS_T(\mathcal{Y}\cup\{X\}) - RSS_T(\mathcal{Y})$ only depends on the values T_t and $\hat{T}_t$ for $t \in X$. Indeed, the models $\mathcal{Y}$ and $\mathcal{Y}\cup\{X\}$ predict the same membership degree $\hat{T}_t$ for $t \notin X$ and the difference of the quadratic residuals over these n-tuples is null. Considering all patterns $X \in \mathcal{X}$, only reading the values T_t and $\hat{T}_t$ for $t \in X$ lowers the time complexity of computing $\arg\min_{X\in\mathcal{X}} RSS_T(\mathcal{Y}\cup\{X\})$ to $O(\sum_{X\in\mathcal{X}} |X|)$.

Algorithm 6: $\arg\min_{X\in\mathcal{X}} RSS_T(\mathcal{Y}\cup\{X\})$

Input: uncertain tensor T, tensor $\hat{T}$ predicted by $\mathcal{Y}$, set $\mathcal{X}$ of patterns ordered by increasing lower-bound
Output: $\arg\min_{X\in\mathcal{X}} RSS_T(\mathcal{Y}\cup\{X\})$

```
1  (min, U) ← (+∞, ∅)
2  while X ≠ ∅ ∧ first lower-bound in X < min do
3      (lower-bound, X) ← X's first entry taken out of X
4      (lower-bound, sum) ← (0, 0)
5      forall t ∈ X s.t. T̂_t < λ_X do
6          term ← (λ_X − T_t)² − (T̂_t − T_t)²
7          if term < 0 then
8              lower-bound ← lower-bound + term
9          sum ← sum + term
10     U ← U ∪ {(lower-bound, X)}
11     if sum < min then
12         (min, argmin) ← (sum, X)
13 forall (lower-bound, X) ∈ U s.t. X ≠ argmin do
14     insert (lower-bound, X) in X respecting the order
15 return argmin
```

However, the worst patterns in $\mathcal{X}$ need not even be considered if information computed in previous iterations is kept in memory. The idea is to store along every pattern $X \in \mathcal{X}$, a lower-bound of $RSS_T(\mathcal{Y}'\cup\{X\}) - RSS_T(\mathcal{Y}')$, where $\mathcal{Y}'$ is any model that can be obtained at any future iteration. In this way, at any future iteration, if the lower-bound associated with a pattern $X \in \mathcal{X}$ exceeds the smallest difference of RSS_T found so far, X cannot be the pattern to select. Given (2.2) and (2.3), if $\hat{T}'_t$ is the membership degree that the future model $\mathcal{Y}'$ predicts for $t \in X$, $RSS_T(\mathcal{Y}'\cup\{X\}) - RSS_T(\mathcal{Y}')$ is:

$$\sum_{t\in X} \left[(\max(\hat{T}'_t, \lambda_X) - T_t)^2 - (\hat{T}'_t - T_t)^2\right] \quad .$$

A lower-bound of that sum is the sum of its negative terms. The term relating to $t \in X$ is negative if and only if:

$$|\max(\hat{T}'_t, \lambda_X) - T_t| < |\hat{T}'_t - T_t| \Leftrightarrow \hat{T}'_t < \lambda_X < 2T_t - \hat{T}'_t \ .$$

The smaller $\hat{T}'_t$ is, the weaker the condition. As a consequence, the number of negative terms is maximal when every $\hat{T}'_t$ is minimal. Moreover, a minimal $\hat{T}'_t$ minimizes the negative term, $(\lambda_X - T_t)^2 - (\hat{T}'_t - T_t)^2$. Indeed, a consequence of the inequality above is $\hat{T}'_t < T_t$. Algorithms 1 and 5 never remove patterns that were previously added to the model. That is why $\mathcal{Y}'$ necessarily is a superset of the model at the current iteration and, given (2.2), $\forall t \in X$, $\hat{T}'_t \geq \hat{T}_t$, where $\hat{T}_t$ is the membership degree that the current model predicts for the n-tuple t. A lower-bound of $RSS_T(\mathcal{Y}' \cup \{X\}) - RSS_T(\mathcal{Y}')$ at the current iteration therefore is:

$$\sum_{t \in X \text{ s.t. } \hat{T}_t < \lambda_X < 2T_t - \hat{T}_t} \left[(\lambda_X - T_t)^2 - (\hat{T}_t - T_t)^2\right] \ .$$

That lower-bound is tight. It is reached when future iterations add to the model patterns at least as dense as X that altogether include every $t \in X$ such that $\lambda_X > 2T_t - \hat{T}_t$ and no pattern that modifies the prediction $\hat{T}_t$ of the membership degree of any $t \in X$ such that $\hat{T}_t < \lambda_X < 2T_t - \hat{T}_t$.

The patterns in $\mathcal{X}$ are stored in a self-balancing binary search tree ordered by increasing lower-bound. The initial bounds, $\sum_{t \in X \text{ s.t. } \lambda_0 < \lambda_X < 2T_t - \lambda_0} [(\lambda_X - T_t)^2 - (\lambda_0 - T_t)^2]$ for $X \in \mathcal{X}$, are computed before the first iteration of Algorithm 1 or Algorithm 5. Algorithm 6 computes $\arg\min_{X \in \mathcal{X}} RSS_T(\mathcal{Y} \cup \{X\})$. Every iteration starts by taking the first pattern X out of $\mathcal{X}$ (line 3). Lines 4–9 compute $RSS_T(\mathcal{Y} \cup \{X\}) - RSS_T(\mathcal{Y})$ and the lower-bound associated with X, given the current model $\mathcal{Y}$. Iterations stop (line 2) when $\mathcal{X}$ is empty or, more commonly, when its first pattern is associated with a lower-bound than is larger than the smallest difference of RSS_T found so far (lines 11–12). Lines 13–14 reinsert in $\mathcal{X}$ the patterns that were taken out of it and whose lower-bounds were updated, except the returned pattern.

Chapter 6

Experimental Validation

`multidupehack` [Cerf and Meira Jr., 2014], `Bigfoot`, `Bigfoot-lr`, PAF and *TriclusterBox* [Mirkin and Kramarenko, 2011] are free software, distributed under the terms of the GNU GPLv3, implemented in C++ and compiled by GCC 5.4.1 with the O2 optimizations. Pauli Miettinen's implementation in C of Walk'n'Merge [Erdös and Miettinen, 2013] bears no license. DBTF [Park et al., 2017] is only distributed in binary form. All experiments were performed on a GNU/Linux™ system running on top of 2.4 GHz cores, 12 MB of cache and 32 GB of RAM. To solve the ILP problems, `Bigfoot` uses the IBM ILOG CPLEX Optimizer v12.6.0 [Bixby et al., 1988]. `https://gitlab.com/lucasmaciel82/Bigfoot` and `https://gitlab.com/lucasmaciel82/PAF` hosts the datasets, the source codes (including for the generation of synthetic tensors) and the scripts to run all experiments.

6.1 Real-World Tensor

6.1.1 Retweets Tensor

The first real-world uncertain tensor used in this experimental chapter is 3-way. It indicates how influential the messages that a Twitter user (among 170,670) wrote about a Brazilian soccer team (among 29, identified by supervised classification) during a week (among 12 in 2014, from January 13$^{\text{th}}$ to April 6$^{\text{th}}$). The influence (i. e., the membership degree) is here defined from how many times the messages were "retweeted" (i. e., republished by other users) and from the overall popularity of the team. More precisely, the numbers of retweets for a given team are multiplied by a constant defined so that the sum of the normalized numbers becomes the average number of retweets per team; a logistic function, with a growth rate of 0.5 and centered on 10 normalized

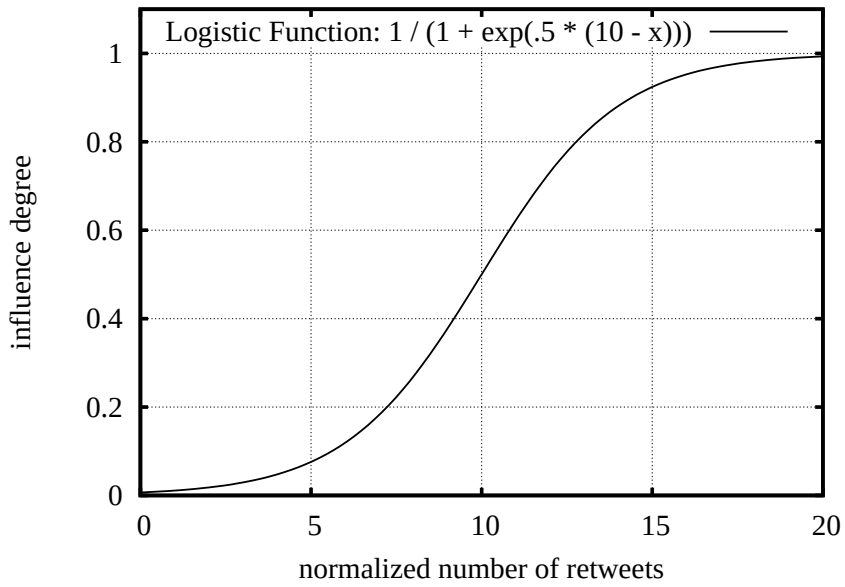


Figure 6.1: The logistic function chosen to turn normalized numbers of retweets into influence degrees: 10 normalized retweets is "moderately influential".

	First Set	Second Set	Third Set
$(\gamma_{\text{users}}, \gamma_{\text{weeks}}, \gamma_{\text{teams}})$	$(8, 3, 4)$	$(8, 2, 4)$	$(2, 4, 3)$
$(\epsilon_{\text{users}}, \epsilon_{\text{weeks}}, \epsilon_{\text{teams}})$	$(3, 8, 6)$	$(0.5, 2, 1)$	$(4, 2, 3)$
Run time (s)	47.4	1.1	147.3
$\lvert\mathcal{F}\rvert$	$1,183,653$	359	$187,617$

Table 6.1: The three sets of initial fragments. The γ is the minimal size constraint where ϵ is the noise tolerated by slice, both respectively for the user, team and week.

retweets (as shown in Figure 6.1), then gives the membership degrees. The resulting $170,670 \times 29 \times 12$ uncertain tensor can be said "sparse": the sum of all its membership degrees is $40,167.3$, i. e., $\frac{40{,}167.3}{170{,}670 \times 29 \times 12} \approx 0.0006$ times the maximal possible value (every membership degree at 1). Since $\lambda_0 = 0.0006$ minimizes $RSS_T(\emptyset)$, 0.0006 can be considered the default value for λ_0, the intercept of the model.

Three different sets of fragments are mined in that tensor. `multidupehack` [Cerf and Meira Jr., 2014] provides them using different minimal size constraints $(\gamma_{\text{users}}, \gamma_{\text{weeks}}, \gamma_{\text{teams}}) \in \mathbb{N}^3$ and different upper-bounds $(\epsilon_{\text{users}}, \epsilon_{\text{weeks}}, \epsilon_{\text{teams}}) \in \mathbb{R}_+$ (see Section 2.1). Table 6.1 lists those parameters as well as the number of fragments `multidupehack` discovers in these settings and the time it takes to do so. The three upper-bounds ϵ_{users}, ϵ_{weeks} and ϵ_{teams} are here actually chosen so that any slice of any minimally-sized fragment has a same minimal density (again, see Section 2.1). For instance, in the First Set, that minimal density is

$\lvert X_{\text{user}}\rvert$	$\lvert X_{\text{weeks}}\rvert$	X_{teams}	**State**	λ_X
31	12	{Botafogo, Flamengo, Fluminense, Vasco}	Rio de Janeiro	0.702
28	11	{Corinthians, Palmeiras, Santos}	São Paulo	0.754
14	10	{Avaí, Figueirense}	Santa Catarina	0.864
17	7	{Grêmio, Internacional}	Rio Grande do Sul	0.859
15	11	{Flamengo, Fluminense}	Rio de Janeiro	0.792
17	12	{Flamengo, Vasco}	Rio de Janeiro	0.801

Table 6.2: The six patterns discovered by `Bigfoot` and PAF in the $170,670 \times 12 \times 29$ Retweet Tensor. PAF returns as well 44 sub-patterns of those six patterns.

$1 - \frac{3}{3\times4} = 1 - \frac{8}{8\times4} = 1 - \frac{6}{8\times3} = 0.75$.

Given that First Set, `Bigfoot-lr` takes 140.4 seconds to turn the $1,183,653$ fragments into only 44 distinct patterns. The high degree of overlap between `multidupehack`'s patterns explains why that number is so small. Feeding `Bigfoot` with those 44 larger fragments and waiting 52 additional seconds allows to discover two patterns. The first pattern is large and rather dense: $\lambda_X = 0.81$. It involves all 12 weeks, 110 influential supporters and journalists (most of them from TV networks) and 13 famous teams, most of them from the Southeast region of Brazil (Corinthians, Flamengo, Cruzeiro, etc.): a total of $17,160$ 3-tuples. The second pattern is much smaller: it contains 150 3-tuples. It is denser though: $\lambda_X = 0.85$. It only involves teams and supporters from the South region of Brazil. `Bigfoot` alone cannot process the $1,183,653$ fragments in a reasonable time. It was aborted after ten hours of unfinished computation. The pre-process and PAF cannot deal with so many fragments and run out of memory.

However, `Bigfoot` and PAF can directly process the Second Set of fragments. By reducing the minimal number of teams in a fragment to two but forcing these fragments to be denser, `Bigfoot` and the forward selection, within 117 minutes, turn those fragments into six patterns. Since there are only 359 fragments, PAF's preprocess is not required. PAF directly agglomerates them. It takes 3.1 seconds to build a dendrogram containing 695 patterns. The forward selection then takes 4.5 seconds to select the same six patterns that `Bigfoot` finds plus 44 sub-patterns of them, that `Bigfoot` cannot find. Table 6.2 lists the six large patterns (first three columns). The fourth column maps the teams involved in the patterns with the state they are in. Indeed, every discovered pattern only involves teams from one single state. The explanation is simple: who is influential when writing about a given team is likely influential when writing about its rivals, in the same state. The last columns of Table 6.2 give the densities of the patterns. The patterns are large and dense.

Table 6.2 lists the patterns in the order they are output, i. e., from the pattern

that most reduces RSS_T to the one that contributes the least to the model. The first four patterns do not intersect, a property favored by the forward selection. They support the discovery of users who were influential when they are writing about all the major teams in a state during most of the weeks. The last two patterns deal with Rio de Janeiro's soccer, which is the most commented. They intersect with each other and with the first pattern in the table. Yet, their addition to the disjunctive box cluster model makes it significantly more accurate (smaller AIC_T). The identifiable users involved in a given pattern are either journalists, who are indeed influential, or supporters of one of the teams in the pattern (Twitter logins including the name or a nickname of the team). The AIC of the model found by `Bigfoot` is $120,156$. It is worse than the model that additionally involve the 44 sub-patterns that PAF returns as well ($AIC_T = 118,434$). PAF is much faster than `Bigfoot` too: 4.5 seconds vs. 117 minutes. That is because PAF's time complexity only depends on the number of fragments at input, whereas `Bigfoot`'s run time depends on the number of tuples in the 46 larger fragments that `Bigfoot-lr` computes.

The fragments in the Third Set involve at least 4 teams. In compensation, the other minimal size constraints are weakened in comparison with the previous settings: at least 2 users and 3 weeks. The fragments in that Third Set tolerate more noise: a minimal possible density of 0.667. With $m = 10^7$, PAF's pre-process grows the $187,617$ fragments into $\sqrt{m} = 3,162$ within 20 minutes and 52s. PAF's run time is 95s. PAF outputs $3,335$ patterns. The forward selection takes 48s and keeps 31 of them, i. e., 6052 times less than the number of initial fragments. The first six patterns are, in average, smaller than those in Tab. 6.2. Many patterns only involve a few weeks. Besides patterns involving rival teams in a same state, four out of the six first patterns stand for teams that played against each other during the weeks in the second dimensions of the patterns. Both `Bigfoot` and `Bigfoot-lr`+`Bigfoot` cannot process the Third Set of fragments in reasonable time.

Since competing algorithms — *TriclusterBox* [Mirkin and Kramarenko, 2011], WALK'N'MERGE [Erdös and Miettinen, 2013] and DBTF [Park et al., 2017] — only handle 0/1 tensors, every membership degree in the uncertain tensor is rounded to 0 or 1. Processing that tensor, *TriclusterBox* times out: within ten hours, it grows one single pattern out of $29 \times 12 = 348$ (see Section 2.2). Despite many attempts with different minimal size and density parameters, WALK'N'MERGE only outputs two patterns within 139 minutes (average run time over the attempts): one very large and sparse pattern involving all weeks, 597 users and 24 teams, and one very dense pattern of size $2 \times 2 \times 2$. DBTF's configuration includes the number of patterns it should discover. However, even when 30 patterns are asked for, DBTF does not find any: it

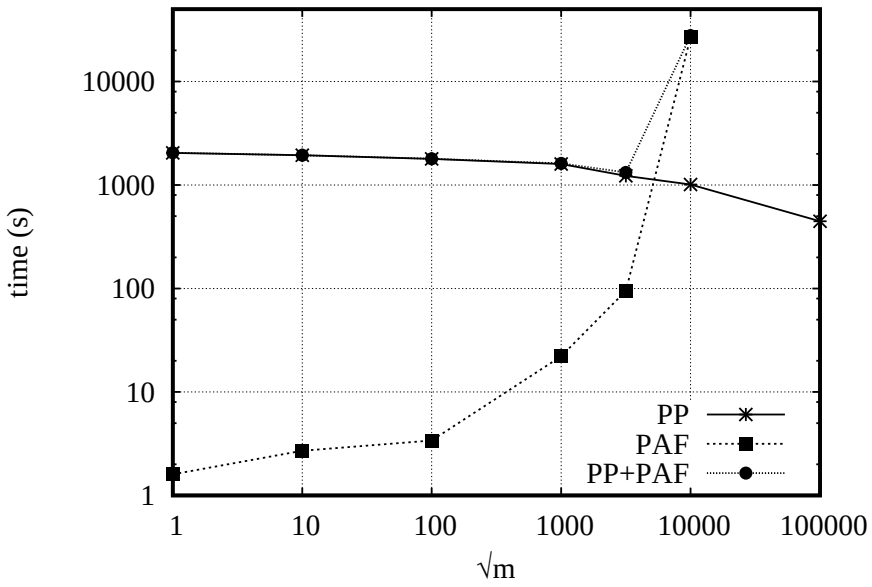


Figure 6.2: Run time of PAF and its pre-process in function of $\sqrt{m}$ varying from 1 to 10^5.

either crashes or returns a factorization of the 0/1 tensor where one of the factor is null.

Influence of m on PAF's performances PAF's pre-process grows the input fragments into at most $\sqrt{m}$ larger fragments. PAF agglomerates them in $O(m)$ time and space. In this section, the parameter m is set to 10^7 ($\sqrt{m} \approx 3,162$). A larger value would make PAF require more than the available 32 GB of RAM. Setting m to a smaller value increases the time and space requirements of the the pre-process, as explained in Section 4.3. Figure 6.2 shows the time PAF and its pre-process take in function of $\sqrt{m}$, varying from 1 to 10^5. The Third Set of fragments (in Table 6.1) is processed here. The AIC of the disjunctive box cluster model obtained with $\sqrt{m} \approx 3,162$ is $322,503$. That configuration provides the lowest total run time among all those that are tested. With $\sqrt{m} \leq 1000$, PAF almost instantly executes but the pre-process requires more time. Moreover, those longer executions return worse models: $AIC_T = 333,760$ for $\sqrt{m} = 1$, $AIC_T = 331,686$ for $\sqrt{m} = 10$, $AIC_T = 327,949$ for $\sqrt{m} = 100$, and $AIC_T = 326,787$ for $\sqrt{m} = 1000$. Overall, the model obtained with $\sqrt{m} = 10^4$ is the best ($AIC_T = 320,875$). On the negative side, it comes at a high computational cost. PAF's agglomeration of $\sqrt{m} = 10^5$ fragments was aborted after ten hours of unfinished computation.

6.1.2 Vélo'v Tensor

The second real-world application deals with the usage of the bicycle sharing network in Lyon, France. That network consists of 327 stations where bicycles can be rented and returned. The riding behaviors evolve along the day and depend on the day of the week. That is why 7×24 directed graphs (one per day of the week and per one-hour period) are built from a two-year log of the system. Their edges are labeled with the numbers of rides between every ordered pair of stations. Those numbers are normalized so that every directed graph has a same total weight. Finally, a logistic function turns every normalized number of rides into a membership degree. The resulting 4-way tensor has a 0.005 density. The intercept of the model, λ_0, is set to that value.

To discover days of the week and periods of these days, during which many users ride between stations in one *single* set, the departure and arrival stations in a pattern must be constrained to be the same. `multidupehack` can enforce that constraint [Cerf and Meira Jr., 2014]. To have `Bigfoot` explore that same restricted pattern space, one additional constraint is added to the ILP problem that f solves at line 12 of Algorithm 1:

$$\forall t \in \mathcal{F}_{\cup},\ x_{t_{\text{departure}}} = x_{t_{\text{arrival}}} \tag{6.1}$$

The hill-climbing procedure has to be modified too: at line 10 of Algorithm 1, one single index stands for both the departure and the arrival stations, i. e., whenever the algorithm considers the addition of a station, it increments both the number of departure and of arrival stations. Those easy changes, which demonstrate the adaptability of the proposal, reduce the time requirements because the pattern spaces explored by hill-climbing and to solve the ILP problem both become smaller. The same modifications are made in PAF's pre-process, at lines 6 and 14. Algorithm 3 doesn't need any modification: given two patterns X and Y that satisfy the constraint 6.1, $X \sqcup Y$ always satisfies that same constraint. Although that particular constraint triggers less modifications in PAF's algorithm than in `Bigfoot`'s, the reverse is usually true: modifying the Integer Linear Programming problem and the hill-climbing procedure to enforce a general constraint is usually easier than modifying PAF's agglomeration.

With $\epsilon_{\text{departure}} = \epsilon_{\text{arrival}} = \epsilon_{\text{hour}} = 5.4$ and $\epsilon_{\text{day}} = 8.1$, `multidupehack` takes 3 minutes and 24s to return $31,509$ maximal cross-graph quasi-cliques (i. e., constrained as 6.1) with at least two days, three one-hour periods and three stations (hence a minimal possible density of 0.7). With $m = 10^7$, PAf's pre-process, PAF itself and the forward-selection turn them into 184 patterns within 8 minutes and 24 seconds.

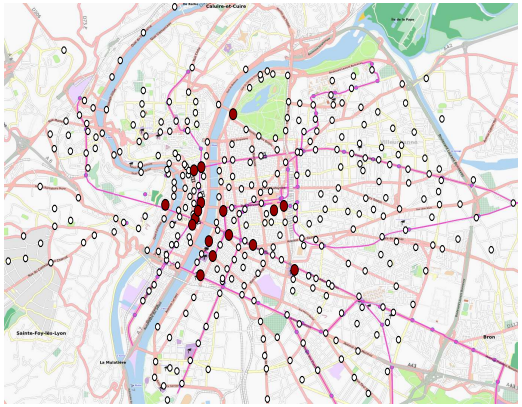
Figures 6.3a, 6.3b, 6.3c and 6.3d show the geographic positions of the stations involved in the first four patterns that are selected, i. e., the four patterns that contribute the most to the model. The captions report the associated days of the week and periods of the day. Background knowledge explaining the patterns is provided as well.

`Bigfoot` and `Bigfoot-lr`+`Bigfoot` cannot process the same 31,509 fragments in reasonable time. However, `Bigfoot` can grow a smaller set of fragments, which cover less 4-tuples: 5,462 fragments, still computed by `multidupehack` with a minimal density of 0.7, but further constrained to involve at least four stations, six one-hour periods and three days. `multidupehack` takes 2 minutes and 44 seconds to return them. `Bigfoot` grows them into two patterns within 1 hour and 48 minutes. The first pattern is similar to the pattern in Figure 6.3a, whereas the second pattern is similar to the one in Figure 6.3d. Together, they make a worse disjunctive box cluster model of the Vélo'v uncertain tensor than the 184 patterns that PAF returns. The available implementations of *TriclusterBox*, Walk'n'Merge and DBTF being restricted to mining 3-way 0/1 tensors, they cannot be used here.

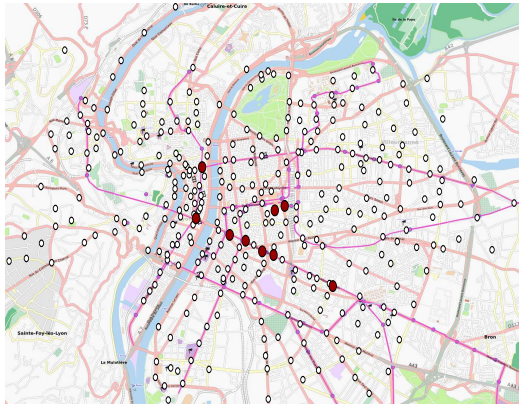
6.2 Synthetic Tensors

The actual patterns to discover in real-world tensors are unknown. To assess to what extent PAF, `Bigfoot` and its variations can recover patterns, this section uses synthetic tensors affected by controlled levels of noise. Four "perfect" patterns (i. e., only containing n-tuples with membership degrees equal to 1) of sizes 8×8 (resp. $6 \times 6 \times 6$) are randomly planted in a null tensor of size 64×64 (resp. $32 \times 32 \times 32$). With those settings, the patterns often overlap, what happens in real-world contexts too, e. g., in Table 6.2 or in Figure 6.3.

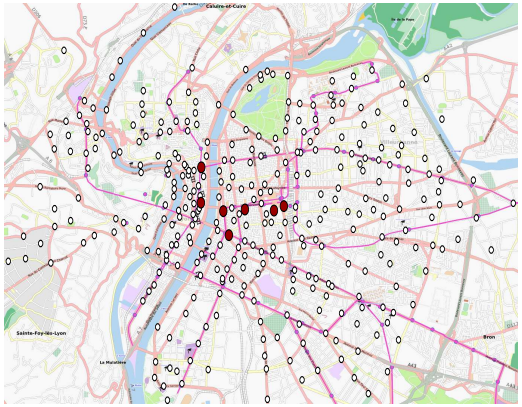
Every 0 or 1 in a "perfect" tensor is then noised by *inverse transform sampling*. To do so, 0 and 1 are considered the output of a Bernoulli variable with parameter p, the probability of a 1. The posterior distribution of p is the beta distribution of parameters α and β. Those two parameters have a meaningful interpretation: after observing, for a same n-tuple, $\alpha - 1$ membership degrees at 1 and $\beta - 1$ at 0, the beta distribution is the distribution of p, the parameter of the Bernoulli variable providing such an output. Choosing a number of correct observations ($\alpha - 1$ when noising a 0, $\beta - 1$ when noising a 1) and a number of incorrect observations ($\beta - 1$ when noising a 0, $\alpha - 1$ when noising a 1) therefore defines the level of noise applied by *inverse transform sampling*. In these experiments, the number of incorrect observations is always set to 0, i. e., only the number of correct observations tunes the level of noise. The greater the



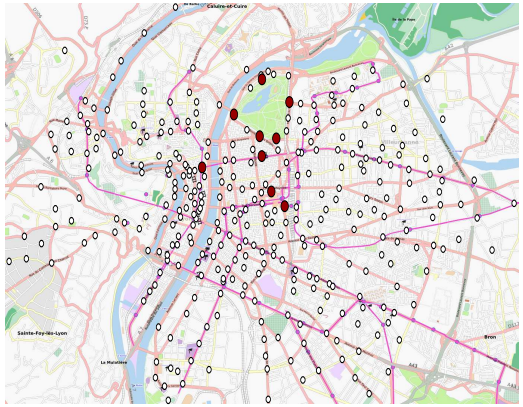
(a) Seventeen large stations in the working districts of Lyon. They exchange many bicycles everyday from midday to 8pm. $|X| = 16184$ and $\lambda_X = 0.239$.



(b) Nine large stations at the main avenue of Lyon. Everyday, from 8am to 9am and from midday to 9pm, many users ride between those stations. $|X| = 5670$ and $\lambda_X = 0.373$.



(c) Seven stations in the commercial districts of Lyon. Users may exchange bicycles everyday, from 8am to 8pm, except on Sundays, when the shops are closed. $|X| = 3234$ and $\lambda_X = 0.460$.



(d) Nine large stations around the main squares of Lyon, in its historical center. Everyday, from 10am to 9pm, many users ride between those stations. $|X| = 6237$ and $\lambda_X = 0.259$.

Figure 6.3: The first four patterns discovered by PAF in the Vélo'v tensor.

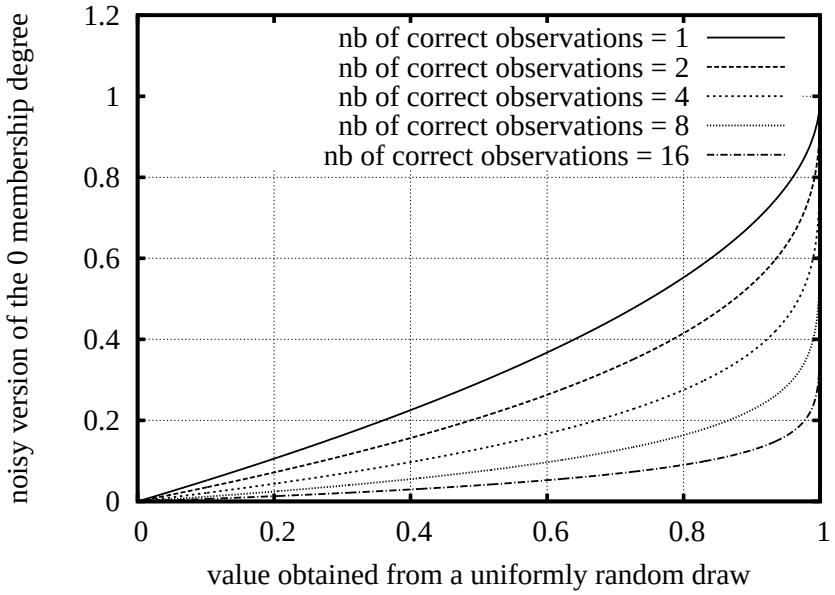


Figure 6.4: Inverses of cumulative beta distributions used to noise a membership degree at 0 in a "perfect" tensor. More "correct observations" mean less noise.

number of correct observations, the less noisy the resulting membership degree, i. e., the closer to the value in the "perfect" tensor. Figure 6.4 plots the inverses of cumulative beta distributions that are used to noise a 0 in a "perfect" tensor: a uniformly random abscissa is drawn in $[0, 1]$ and the related ordinate is read on the curve whose number of correct observations provides the desired level of noise. That ordinate is the noisy version of the 0 in the "perfect" tensor.

Given a planted pattern P and a pattern X at the output of a method, the Jaccard index $J(P, X)$ measures their similarity:

$$J(P, X) = \frac{|P \cap X|}{|P \cup X|} \ .$$

Given $\mathcal{P}$, the planted patterns, and $\mathcal{X}$, the patterns discovered in the related uncertain tensor, the *quality* of $\mathcal{X}$ is computed in this way:

$$\frac{|\bigcup_{P \in \mathcal{P}} (P \cap \arg\max_{X \in \mathcal{X}} J(P, X))|}{|\bigcup_{P \in \mathcal{P}} P \cup \bigcup_{X \in \mathcal{X}} X|} \ .$$

That quality measure, in $[0, 1]$, is a ratio between numbers of n-tuples. At the numerator, the true positive n-tuples are both in a planted pattern and in the pattern of $\mathcal{X}$ that is the most similar. The denominator is the number of n-tuples in the planted patterns or in the discovered patterns. The quality measure penalizes both the ab-

sence from $\mathcal{X}$ of patterns that are similar to the planted ones and the discovery of patterns including n-tuples out of the planted patterns. It does not take into account the possible redundancy of information in $\mathcal{X}$, i. e., does not penalize the discovery of supernumerary overlapping patterns. That is why the following experimental results report as well the number of patterns that are discovered, $|\mathcal{X}|$.

Figures 6.5, 6.6 and 6.7 respectively plot the quality of the discovered patterns, their numbers, and the run times in function of the level of noise in the uncertain tensor. For a given level of noise, all results are averages over eight randomly generated tensors. `multidupehack` [Cerf and Meira Jr., 2014] provides the initial fragments, with at least four elements of every dimension (respectively three elements in 3-way tensors), i. e., n minimal size constraints are enforced. Different upper-bounds $(\epsilon_1, \ldots, \epsilon_n)$ are tested so that the initial fragments can be more or less dense. The upper-bound is always the same whatever the dimension: $\exists \epsilon \in \mathbb{R}_+ \mid \forall i \in \{1, \ldots, n\}$, $\epsilon_i = \epsilon$. That upper-bound ϵ relates to μ, given at the top of the figures: $\epsilon = \gamma^{n-1}(1 - \mu)$. For instance, for $n = 3$, $\gamma = 3$ and $\mu = 0.6$ (bottom-right plots), `multidupehack`'s upper-bounds are set to $(3.6, 3.6, 3.6)$. Given `multidupehack`'s definition of a fragment and the minimal size constraints, $\mu = 1 - \frac{\epsilon}{\gamma^{n-1}}$ is the minimal possible density for any slice of a fragment. Here, the intercept λ_0 is always set to 0. Forward selection (see Chapter 5) is used to simplify the output of all methods but `multidupehack`'s, so that the overall gains can be observed. Whatever the post-processed method, the forward selection takes a negligible time, improves the quality and, of course, decreases the number of patterns.

`multidupehack` returns large collections of fragments (Figure 6.6) that poorly match the planted patterns (Figure 6.5), unless the level of noise is low (16 correct observations). Nevertheless, PAF, `Bigfoot` and `Bigfoot-lr`+`Bigfoot`, which both process `multidupehack`'s outputs, reach significantly higher qualities after the forward selection that keeps numbers of patterns that are close to four, the number of planted patterns. PAF is almost always the best performer, followed by `Bigfoot`. That is, when it is actually given fragments ($\mu = 0.8$ is too high for `multidupehack` to return any fragment in the noisiest tensors) and when it is not given too many fragments, what makes its run time the chosen timeout, 1 hour. That happens when processing the sparse fragments ($\mu = 0.6$) in the noisiest matrices ($n = 2$. In that setting, PAF times out as well. The differences between the two proposals are clearer when they are applied on uncertain matrices ($n = 2$). In that context, `Bigfoot` more often finds patterns that go over edges of the planted patterns, as explained at the end of Section 3.1. When PAF processes the fragments obtained from the 3-way tensors with $\mu = 0.7$, it almost always ends up returning the four patterns that were planted. It takes less than 10 seconds (including the run time of the pre-process; m is here set to

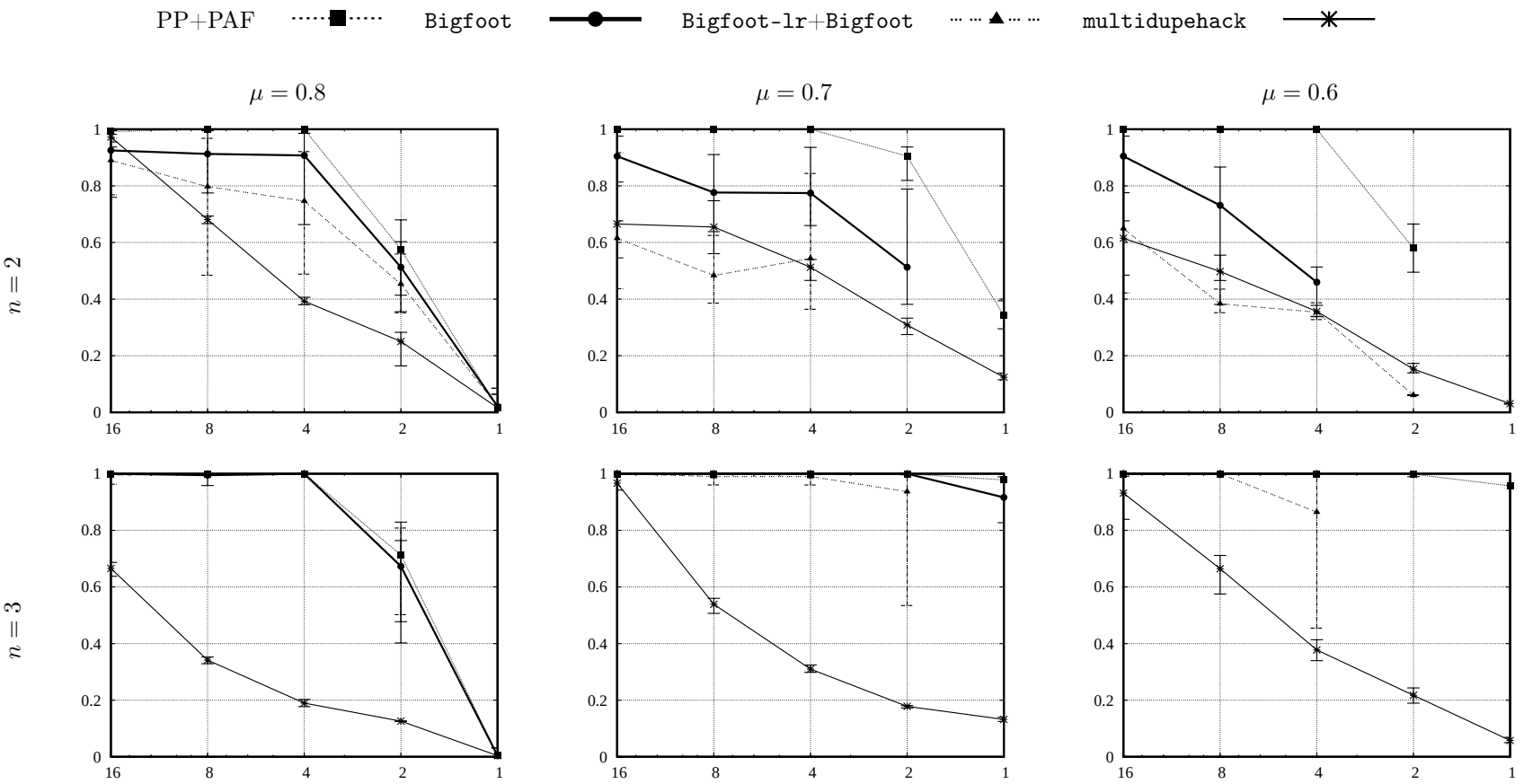


Figure 6.5: Qualities of the patterns discovered by the different methods in function of the number of correct observations (the noise increases from left to right).

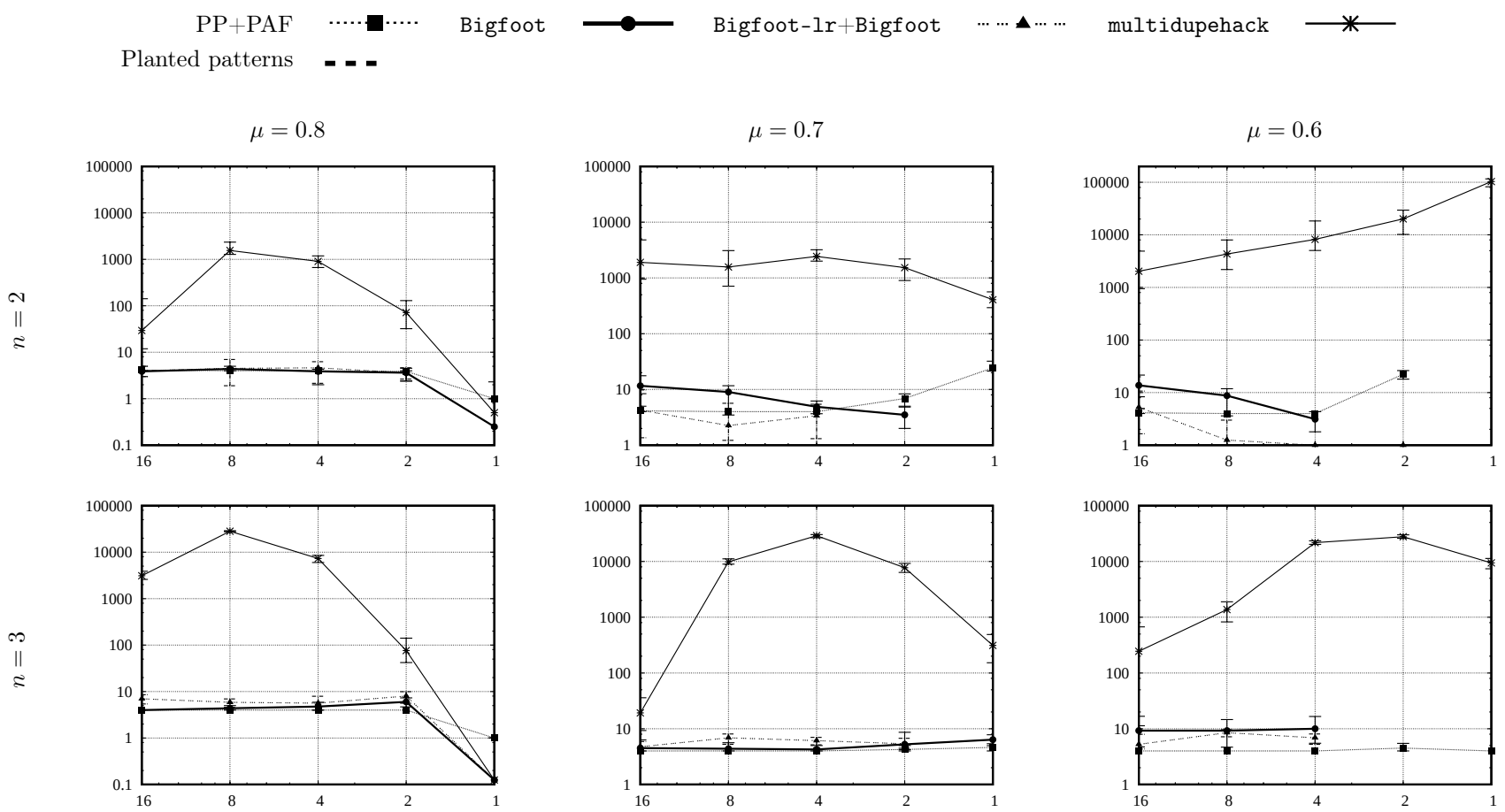


Figure 6.6: Numbers of patterns discovered by the different methods in function of the number of correct observations (the noise increases from left to right).

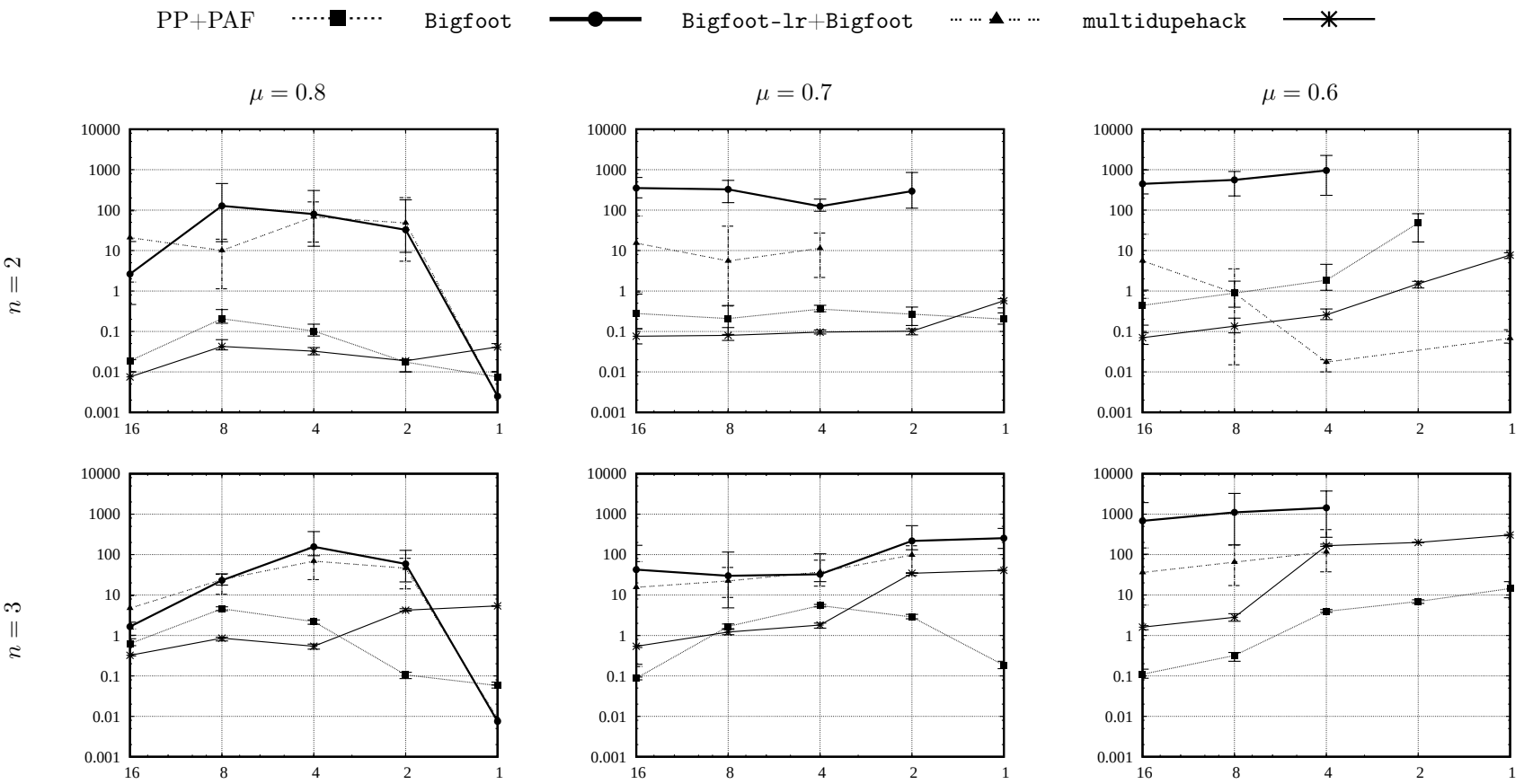


Figure 6.7: Run times (in seconds) of the different methods in function of the number of correct observations (the noise increases from left to right). `mutidupehack`'s execution is included in the times reported for PAF, `Bigfoot` and `Bigfoot-lr+Bigfoot`.

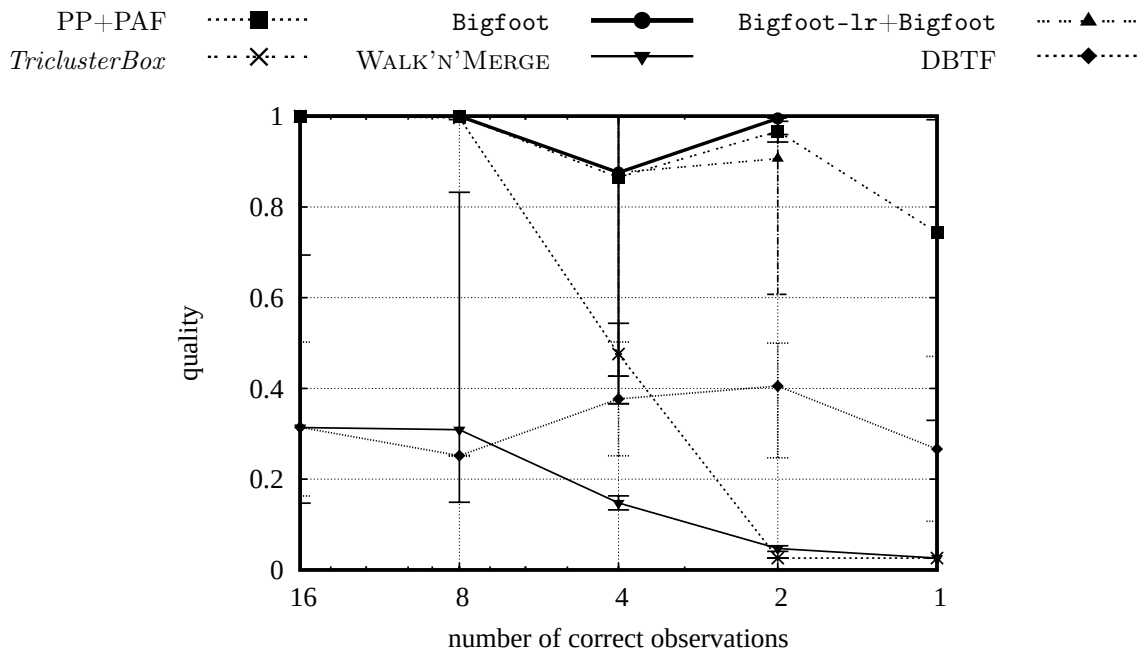


Figure 6.8: Qualities of the patterns discovered in the 3-way rounded tensors by the different methods in function of the number of correct observations (the noise increases from left to right).

10^6) to do so.

`Bigfoot-lr`+`Bigfoot`'s patterns are of high quality too. They are not as good as PAF and `Bigfoot`'s though. `Bigfoot-lr`+`Bigfoot` usually runs faster than `Bigfoot`, sometimes much faster. It is the case when processing fragments of low density ($\mu = 0.6$) in the matrices that were synthesized. However, `Bigfoot-lr`+`Bigfoot` can be slower than `Bigfoot` too, as explained in Section 3.3. It happens for instance in tensors with 2 correct observations when processing fragments obtained with $\mu = 0.8$.

How dense the fragments, provided by `multidupehack`, alters not only the run times of the algorithms but also the qualities of the disjunctive box cluster model they return. The best results are here observed with $\mu = 0.7$. With $\mu = 0.6$, `multidupehack` returns many fragments and, in the noisiest uncertain tensors, some of them go over edges of the planted patterns. On the contrary, with $\mu = 0.8$, there are too few fragments: they do not include all the n-tuples in the planted patterns.

PAF and `Bigfoot` (and its variations) handle n-way uncertain tensors, whereas the state-of-art only deals with 3-way 0/1 tensors. In that context, Figures 6.8, 6.9 and 6.10 show the qualities, the numbers of patterns and the run times of PAF, `Bigfoot`, `Bigfoot-lr`+`Bigfoot`, *TriclusterBox*, WALK'N'MERGE and DBTF. `multidupehack` provides the fragments, still with at least three elements per dimension, that PAF's

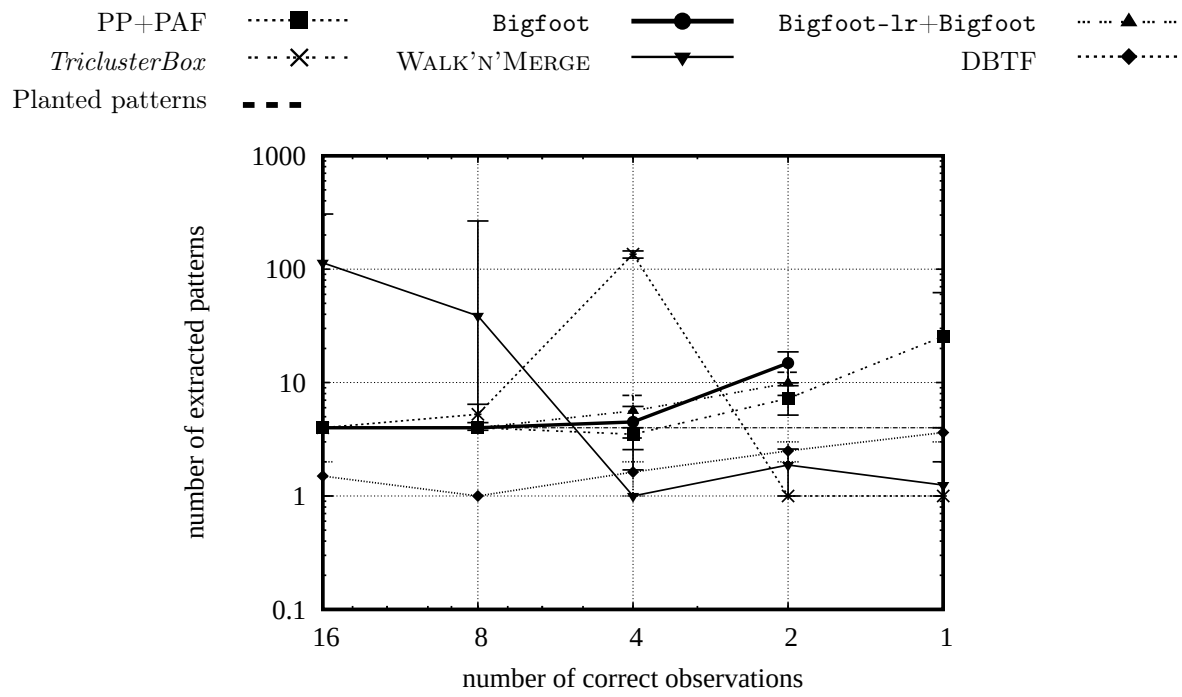


Figure 6.9: Number of patterns discovered in the 3-way rounded tensors by the different methods in function of the number of correct observations (the noise increases from left to right).

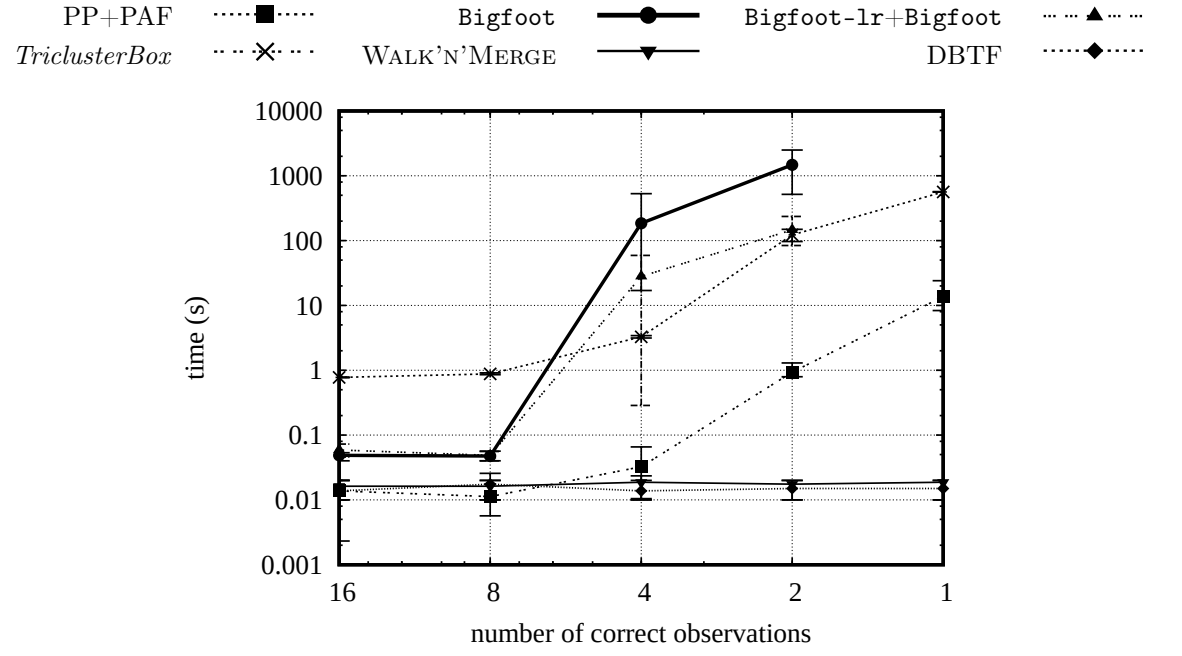


Figure 6.10: Running times of the the different methods in the 3-way rounded tensors in function of the number of correct observations (the noise increases from left to right).

pre-process, `Bigfoot` and `Bigfoot-lr`+`Bigfoot` grow. Its upper-bounds $(\epsilon_1, \epsilon_2, \epsilon_3)$ are set to $(3, 3, 3)$, i. e., any slice of any fragment contains at most three 3-tuple with a null membership degree. That corresponds to $\mu = 0.7$ in the previous experiments. In Figure 6.8, PAF and `Bigfoot` reach qualities that are consistently lower than those in Figure 6.5 with $n = 3$ and $\mu = 0.7$: rounding the membership degrees really harms the ability to recover the planted patterns. Despite the damage that rounding causes, PAF, `Bigfoot` and, to a lesser extent, `Bigfoot-lr`+`Bigfoot`, are clearly better than *TriclusterBox*, WALK'N'MERGE and DBTF at recovering the planted patterns from the noisy 0/1 tensors. WALK'N'MERGE's minimal density is set to 0.7 and all other parameters are set to their default values. Other configurations were tested but they did not yield better results. Contrary to WALK'N'MERGE, which uses the Minimal Description Principle to not overfit the tensor, *TriclusterBox* includes no such mechanism. The forward selection, described in Chapter 5, always improves *TriclusterBox*'s output. Its results in Figures 6.8, 6.9 and 6.10 include that step. The number of patterns DBTF should discover is part of its configuration. Although that number is set to the number of planted patterns, four, DBTF usually returns less patterns. DBTF's authors denying access to its source code, that bug could not be fixed.

The experiments on synthetic tensors confirm what those on real-world tensors (in Section 6.1) suggest: PAF, with its pre-process, is the best algorithm to summarize uncertain tensors with patterns and their densities. Not only the returned disjunctive box cluster models are high-quality, even when the tensor is rather noisy, but the parameter m allows to quite easily tune the run time and the memory consumption. `Bigfoot` sorely lacks that second property. Its run time depends on the (unpredictable) number of iterations the hill-climbing procedures require to reach the local maximums of g and the times to solve the ILP problems may exponentially grow with these numbers of iterations. Although `Bigfoot` (and `Bigfoot-lr`+`Bigfoot`) have poorer performance than PAF, the disjunctive box cluster models they return are almost good as those computed with PAF. Not only, the current state-of-art algorithms cannot mine *uncertain* tensors but the experiments show they are clearly worse than the proposal in this dissertation when it comes to recovering patterns in noisy 0/1 tensors.

Chapter 7

Conclusion

Discovering a disjunctive box cluster model is a problem that generalizes the Boolean CP tensor factorization: every pattern (rank-1 tensor) is weighted by a parameter to estimate. Searching for patterns one by one, the optimal weights simply are their densities. Such a disjunctive box cluster model therefore is more informative than a Boolean CP factorization but it remains easy to interpret. Moreover, it suits uncertain tensors and not only 0/1 tensors.

This work has presented two solutions to decompose uncertain tensors into patterns. Both of them include an optional step to reduce the set of fragments with pre-processing procedures. PAF's pre-processing grows each fragment using the RSS_T difference while `Bigfoot-lr` uses a relaxation of the ILP model. After, `Bigfoot` grows each fragment by hill-climbing while PAF agglomerates them building a dendrogram that summarizes the tensor in a better structured way. Finally, forward selection composes the returned model, a subset of the discovered patterns, by greedily minimizing the AIC. Experiments have shown that the methods successfully recover patterns in noisy synthetic tensors. They even outperform state-of-the-art approaches when the tensor is 0/1, a special case. Relevant patterns have been found in two real-world uncertain tensors with tens of millions of values.

The contributions of this work are not limited to the results of the proposals, but on proposals itself. Some unusual techniques in pattern mining as integer linear programming are explored and showed be useful and promising. The integer linear model defined for `Bigfoot` enabled maximizing the integer non-linear function of the disjunctive box cluster model by maximizing multiple times a much easier integer linear function. Its relaxation was introduced as well. Future improvement on `Bigfoot` is changing the solver to a more specific one, mixing a branch-and-cut technique with the hill-climbing. The idea is to use the branch-and-cut to maximize the integer linear

model proposed and grow the cardinalities of the pattern dimensions when reach to a local maximum in the linear function.

The output of PP+PAF (PAF and its pre-processing procedure), which have shown the highest quality results and performance, can be better explored and visualized as well. The dendrogram carries more information from the tensor than just the patterns that can be useful for an analyst. However, showing a tree as in Figure 1.2 is unpractical since the dendrogram can be large, although, a better solution should be proposed in a future work.

Both `Bigfoot` and PAF (actually, the PAF's pre-processing) could have their run times significantly reduced if we distribute their computation. In case of `Bigfoot`, a branch-and-cut method is easy to parallelize since each branch could be computed separately. PAF's pre-processing as well as `Bigfoot` can be computed in parallel: dividing the initial set of fragments in portions and computing PAF's pre-processing in each of them. However it may produce a different output since it would change the growing order of each fragment. Other distributed algorithms can be proposed to pre-processing the fragments. To assert the performance and the quality of the distributed methods suggested, new experiments are demanded in the next future works.

A big contribution to pattern mining is creating a framework and makes it available to everyone. Within `multidupehack`, PAF, `Bigfoot`, and the forward selection the framework should receive any uncertain tensor and return the patterns of interest the analyst (user) is looking for. Since `multidupehack` requires a previous knowledge of the user from the dataset to set correctly the minimum size and density constraints, a future improvement is to use the Minimum Description Length principle techniques to predicts the set of `multidupehack`'s parameters that makes it find a good initial set of fragments. This feature will make the framework even more transparent and accessible. A robust version of the framework could have a better visualization of the patterns and the distributed version of the algorithms as suggested in the previous paragraphs. Finally, all other approaches that deal with uncertain tensors, numeric tensors or 0/1 tensors distributed under the terms of the GNU GPLv3 can be part of the framework.

Bibliography

Akaike, H. (1974). A new look at the statistical model identification. *Trans. on Automatic Control*, 19(6):716--723.

Bixby, R. E., ILOG, and IBM (1988). Cplex optimizer.

Bělohlávek, R., Glodeanu, C., and Vychodil, V. (2013). Optimal factorization of three-way binary data using triadic concepts. *Order*, 30(2):437--454.

Cerf, L., Besson, J., Robardet, C., and Boulicaut, J.-F. (2009a). Closed patterns meet n-ary relations. *ACM Trans. on Knowl. Disc. from Data*, 3(1):1--36.

Cerf, L. and Meira Jr., W. (2014). Complete discovery of high-quality patterns in large numerical tensors. In *Proc. of the Int. Conf. on Data Eng.*, pages 448--459.

Cerf, L., Mougel, P.-N., and Boulicaut, J.-F. (2009b). Agglomerating local patterns hierarchically with ALPHA. In *Proc. of the ACM Conf. on Inf. and Knowl. Management*, pages 1753--1756.

Cheng, H., Yu, P. S., and Han, J. (2006). Ac-close: Efficiently mining approximate closed itemsets by core pattern recovery. In *ICDM'06: Proceeding of the Sixth International Conference on Data Mining*, pages 839--844. IEEE Computer Society.

Dummit, D. S. and Foote, R. M. (2003). *Abstract Algebra, 3rd Edition*. Wiley.

Erdös, D. and Miettinen, P. (2013). Walk 'n' merge: A scalable algorithm for boolean tensor factorization. In *ICDM'13: Proceedings of the 13th International Conference on Data Mining*, pages 1037--1042. IEEE Computer Society.

Erwin Abbink, L. G. e. a. (2017). *Optimization in the Railway Industry*. Springer International Publishing.

Georgii, E., Tsuda, K., and Schölkopf, B. (2011). Multi-way set enumeration in weight tensors. *Machine Learning*, 82(2):123--155.

Grünwald, P. D. (2007). *The Minimum Description Length Principle*. The MIT Press.

Gunjan K. Gupta, A. S. and Ghosh, J. (1999). Distance based clustering of association rules. In *ANNIE '99: Proceedings of the 9th Intelligent Engineering Systems Through Artificial Neural Networks*, pages 759--764. ASME Press.

Hannu Toivonen, Mika Klemettinen, P. R., Hätönen, K., and Mannila, H. (1995). Pruning and grouping discovered association rules. In *Proceedings of the ECML '95 Workshop on Statistics, Machine Learning and Knowledge Discovery in Databases*, pages 47--52.

Ignatov, D. I., Gnatyshak, D. V., Kuznetsov, S. O., and Mirkin, B. G. (2015). Triadic formal concept analysis and triclustering: Searching for optimal patterns. *Machine Learning*, 101(1):271--302.

Leenen, I., Mechelen, I. V., Boeck, P. D., and Rosenberg, S. (1999). INDCLAS: A three-way hierarchical classes model. *Psychometrika*, 64(1):9--24.

Meindl, B. and Templ, M. (2013). Analysis of commercial and free and open source solvers for the cell suppression problem. pages 147--159. Transactions on Data Privacy.

Metzler, S. and Miettinen, P. (2015). Clustering boolean tensors. *Data Mining and Knowl. Disc.*, 29(5):1343--1373.

Miettinen, P. (2011). Boolean tensor factorizations. In *ICDM'11: Proceedings of the 11th International Conference on Data Mining*, pages 447--456. IEEE Computer Society.

Mirkin, B. G. and Kramarenko, A. V. (2011). Approximate bicluster and tricluster boxes in the analysis of binary data. In *Proc. of the Int. Conf. on Rough Sets, Fuzzy Sets, Data Mining and Granular Computing*, pages 248--256.

Nemhauser, G. L. and Wolsey, L. A. (1988). *Integer and combinatorial optimizations*. Wiley.

Park, N., Oh, S., and Kang, U. (2017). Fast and scalable distributed boolean tensor factorization. In *Proc. IEEE International Conference on Data Engineering (ICDE'17)*, pages 1071--1082, San Diego, CA, USA.

Poernomo, A. K. and Gopalkrishnan, V. (2007). Mining statistical information of frequent fault-tolerant patterns in transactional databases. In *ICDM'07: Proceedings of the Seventh International Conference on Data Mining*, pages 272--281. IEEE Computer Society.

Poernomo, A. K. and Gopalkrishnan, V. (2009). Towards efficient mining of proportional fault-tolerant frequent itemsets. In *KDD'09: Proceedings of the 15th SIGKDD international conference on Knowledge discovery and data mining*, pages 697--706. ACM Press.

Rokach, L. and Maimon, O. (2005). *Data mining and knowledge discovery handbook*. Springer US.

Schrijver, A. (1998). *Theory of linear and integer programming*. John Wiley and Sons.

Spyropoulou, E. and Bie, T. D. (2014). Mining approximate multi-relational patterns. In *Proc. of the Int. Conf. on Data Science and Advanced Anal.*, pages 477--483.

Spyropoulou, E., Bie, T. D., and Boley, M. (2014). Interesting pattern mining in multi-relational data. *Data Mining and Knowl. Disc.*, 28(3):808--849.

Sylvain Blachon, Ruggero G. Pensa, J. B., Robardet, C., Boulicaut, J.-F., and Gandrillon, O. (2007). Clustering formal concepts to discover biologically relevant knowledge from gene expression data. *In Silico Biology*, 7(33):1--15.

Wong, A. K. C. and Li., G. C. L. (2008). Simultaneous pattern and data clustering for pattern cluster analysis. *IEEE Transations on Knowledge and Data Engineering*, 20(7):911--923.

Wong, A. K. C. and Li, G. C. L. (2008). Simultaneous pattern and data clustering for pattern cluster analysis. *Trans. on Knowl. and Data Eng.*, 20(7):911--923.

Zhao, L. and Zaki., M. J. (2005). Microcluster: Efficient deterministic biclustering of microarray data. *IEEE Intelligent Systems*, 20(6):40--49.

Zhao, L. and Zaki, M. J. (2005). TRICLUSTER: An effective algorithm for mining coherent clusters in 3D microarray data. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 694--705.