International Journal of Computational Intelligence Systems
Vol. **13(1)**, 2020, *pp.* 1036–1047
DOI: https://doi.org/10.2991/ijcis.d.200715.002; ISSN: 1875-6891; eISSN: 1875-6883
https://www.atlantis-press.com/journals/ijcis/

Research Article

Climbing the Hill with ILP to Grow Patterns in Fuzzy Tensors

Lucas Maciel, Jônatas Alves, Vinicius Fernandes dos Santos, Loïc Cerf *,

Computer Science Department, Federal University of Minas Gerais, Avenida Antônio Carlos 6627 - Prédio do ICEx, Belo Horizonte, Minas Gerais 31270-901, Brazil

ARTICLE INFO

Article History

Received 08 May 2020
Accepted 13 Jul 2020

Keywords

Disjunctive box cluster model
Fuzzy tensor
Hill-climbing
Integer Linear Programming
Forward selection

ABSTRACT

Fuzzy tensors encode to what extent n-ary predicates are satisfied. The *disjunctive box cluster model* is a regression model where sub-tensors are explanatory variables for the values in the fuzzy tensor. In this article, locally optimal patterns for that model, with high areas times squared densities, are grown by hill-climbing from fragments of them. A *forward selection* then chooses among the discovered patterns a non-redundant subset that fits, but does not overfit, the tensor.

1. INTRODUCTION

Suppose an analyst wants to study the types of chocolate consumers like depending on the regions they live in. If she has the results of a survey where customers from different regions grade the different types of chocolate, she can scale those grades to $[0, 1]$, where 0 stands for "absolutely hating" and 1 for "absolutely loving," and compute averages per region. Table 1 shows a matrix our analyst may end up with. Since any of its elements is necessarily in $[0, 1]$, the matrix is said *fuzzy* [1].

This article details the first method to fit a *disjunctive box cluster model* to an n-way *fuzzy tensor*. A fuzzy *matrix*, such as in Table 1, is a special case: $n = 2$. The disjunctive box cluster model, that Mirkin and Kramarenko introduced in 2011 for 0/1 tensors [2], is a pattern-based summary of the data. In that model, a pattern is a sub-tensor. It is associated with a value in $[0, 1]$: its density, *i.e.*, the arithmetic mean of the values in the sub-tensor. Table 1 emphasizes a pattern (all grayed cells) whose density is $\frac{9.19}{12} \approx 0.77$. Its interpretation is easy: the consumers in the Southeast, North and West-Center regions like bitter, crunchy, semisweet and sweet chocolates to a certain degree, 0.77, the average grade these three regions give to these four chocolates. In addition to that pattern, the proposal discovers {South, Southeast, Northeast} × {milky, sweet, white} of density $\frac{7.68}{9} \approx 0.85$. Together they summarize the fuzzy matrix. Table 2 shows the approximate reconstruction of the matrix from the model. Every value is, here, assumed null (the analyst can actually specify a different value) unless the related cell belongs to some pattern. If so, the model predicts that the value is the density of the densest pattern containing the cell. Every pattern density is indeed seen as the *truth value* (not the probability) of the predicate "the regions (the rows of the pattern) like the chocolates (its columns)" and, the disjunctive box cluster model being disjunctive, Zadeh's OR operator (*i.e.*, the max operator) is used.

Table 1 A fuzzy matrix, a pattern (all grayed cells) and a fragment of this pattern (darker cells).

	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

The disjunctive box cluster model is actually a regression model: the patterns and their densities *explain* the values in the fuzzy tensor. Using ordinary least squares, fitting such a model to a fuzzy tensor aims to minimize the sum of the squared differences between every value in the fuzzy tensor (*e.g.*, Table 1) and the analog value in the predicted tensor (*e.g.*, Table 2). In that framework, Mirkin and Kramarenko prove [2] that the pattern that, alone, best summarizes the tensor is the one with the greatest area (the number of cells it contains) times density squared [2], hence a statistically founded formalization of the usual desire, in the pattern-mining community, to discover large and dense patterns in fuzzy (or simply 0/1) datasets.

To discover patterns that are good explanatory variables for a disjunctive box cluster model, the present article proposes to first run an existing algorithm that provides many small high-density patterns. They are here called *fragments* because the second step grows each of these fragments, one by one, by hill-climbing in the space of the cardinalities of the pattern dimensions, until reaching a pattern having a locally maximal area times density squared. For example,

* *Corresponding author. Email: lcerf@dcc.ufmg.br*

Table 2 Fuzzy matrix predicted by the disjunctive box cluster model composed of the two highlighted patterns of densities $\frac{9.19}{12} = 0.77$ and $\frac{7.68}{9} = 0.85$ (predicted at the intersection of the two patterns for being greater).

	Bitter	Crunchy	Milky	Semisweet	Sweet	White
South	0	0	0.85	0	0.85	0.85
Southeast	0.77	0.77	0.85	0.77	0.85	0.85
North	0.77	0.77	0	0.77	0.77	0
Northeast	0	0	0.85	0	0.85	0.85
West-Center	0.77	0.77	0	0.77	0.77	0

the fragment of size 2×3 in Table 1 grows into its super-pattern of size 3×3 or 2×4 with the greatest area times density squared. Assuming it is a 3×3-pattern, the algorithm then considers patterns of size 4×3 or 3×4. To more thoroughly explore the pattern space, they are only required to be super-patterns of the initial fragment and not necessarily super-patterns of the pattern chosen at the previous iteration. When a local maximum is reached, the related pattern becomes a candidate explanatory variable for the disjunctive box cluster model. Finally, by greedily minimizing the Akaike Information Criterion [3], a stepwise regression technique selects a subset of those candidate variables that fits but does not overfit the fuzzy tensor.

The main contributions of this article are

- The disjunctive box cluster model is shown to generalize the Boolean CANDECOMP/PARAFAC (CP) decomposition;
- The first pattern-based method to decompose *fuzzy* tensors is presented (existing approaches need to round the values in $[0, 1]$ to 0 or 1, a loss of information);
- Experiments show it discovers high-quality patterns in fuzzy tensors and outperforms state-of-the-art algorithms when applied to 0/1 tensors, a special case.

After a few basic definitions in Section 2, Section 3 discusses the related work. In particular, it presents the disjunctive box cluster model. Section 4 details the proposal to fit (but not overfit) such a model to a fuzzy tensor. Section 5 shows the proposal successfully identifies patterns in real-world and synthetic fuzzy tensors and outperforms existing algorithms when the tensor is 0/1. Section 6 concludes the paper.

2. BASIC DEFINITIONS

Given $n \in \mathbb{N}$ *dimensions* (*i.e.*, n finite sets, assumed disjoint w.l.o.g.) $D_1, \cdots, D_n$, a *fuzzy 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. A set of n-tuples $X \subseteq \prod_{i=1}^{n} D_i$ is a *pattern* if and only if $\forall i \in \{1, \ldots, n\}, \exists X_i \subseteq D_i$ 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 $X \subseteq Y$.

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 set of n-tuples $S \subseteq \prod_{i=1}^{n} D_i$ and an element $e \in D_i$, $\{t \in S \mid t_i = e\}$ is a *slice* of S. If $n = 2$, the slices of a pattern are its rows and columns.

3. RELATED WORK

The above definition of a pattern is purely syntactical. To be semantically *relevant*, a pattern must contain a "large" number of n-tuples and their membership degrees must be "high." The data-mining literature formalizes and fulfills those objectives in different ways. This section focuses on patterns in 0/1 and fuzzy n-way tensors with $n \geqslant 3$. However, the next paragraph presents algorithms that only handle 0/1 matrices (*i.e.*, $n = 2$) but that technically relate to the present proposal.

3.1. Complete Algorithms

Given a 0/1 matrix, AC-Close [4] grows its all-ones sub-matrices, mined in a preprocessing step, into patterns with *dense* rows and columns: the proportion of 1 in every row must exceed a user-defined minimal density threshold and so does the proportion of 1 in every column (but the threshold may be different). Those patterns are forced to be *closed* too: any super-pattern has at least one row or column that is not dense enough. Poernomo and Gopalkrishnan use Integer Linear Programming (ILP) to output the same type of pattern [5] or a similar type of pattern [6] where the *absolute* number of 0 in any row/column is upper-bounded.

Given a 0/1 tensor, several algorithms, *e.g.*, Data-Peeler [7], list its all-ones sub-tensors that are *closed*, *i.e.*, any strict super-pattern includes an n-tuple with a null membership degree. Ignatov *et al.* survey relaxations of that definition [8]. RMiner [9] generalizes Data-Peeler's definition to a multi-relational context: 0/1 tensors sharing dimensions. RMiner's patterns do not tolerate n-tuples with null membership degrees. In contrast, A-RMiner's patterns [10] do, by agglomeration of patterns that RMiner computes and that share a common core. Given prior beliefs, both proposals [9,10] deem a pattern interesting if the ratio between its information content, which grows with its area, and its description length, which grows with its number of elements in all dimensions, is high.

multidupehack [11] generalizes Data-Peeler to fuzzy tensors: n upper-bounds, $(\epsilon_1, \ldots, \epsilon_n) \in \mathbb{R}_+^n$, specify how much the sum of the membership degrees of the n-tuples in every slice of a pattern can deviate from the one that would be obtained if these membership degrees were all 1. Formally, multidupehack outputs the pattern $X = \prod_{i=1}^{n} X_i$ if $\forall i \in \{1, \ldots, n\}$, $\forall e \in X_i$, $\sum_{t \in X \text{ s.t. } t_i = e} (1 - T_t) \leqslant \epsilon_i$. In that definition, and considering Zadeh's NOT operator, $1 - T_t$ quantifies to what extent the n-tuple t is absent from the fuzzy n-ary relation T encodes. In this way, the threshold $\epsilon_i \in \mathbb{R}_+$ upper -bounds the number of absent n-tuples in any slice of the pattern where the fixed component e of the n-tuples is taken in X_i. multidupehack, like Data-Peeler, enforces as well a closedness constraint, which discards all strict sub-patterns of a valid pattern, and can prune the search with additional relevance constraints that every pattern must satisfy, *e.g.*, the minimal size constraint — "involving at least $\gamma_i \in \mathbb{N}$ elements of D_i."

Dense Cluster Enumeration (DCE) [12] is the only other complete algorithm to mine patterns in fuzzy tensors. Cerf and Meira show that DCE's definition catches patterns that are not sub-patterns of any pattern planted in a synthetic dataset, even if little noise affects that dataset [11]. In contrast, multidupehack's patterns do not go over the edges of the planted patterns, unless the tensor is

very noisy. Moreover, multidupehack scales better than DCE. Yet, increasing its bounds $\epsilon_1, \cdots, \epsilon_n$ 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.

3.2. Heuristic Algorithms

TRICLUSTER [13] mines patterns in 3-way tensors. It can optionally postprocess them to reach larger patterns that tolerate more noise: two overlapping patterns are merged into the smallest pattern containing both if there is a large-enough ratio between the number of 3-tuples in either pattern and the number of 3-tuples in neither of them but in the merged pattern. That process ignores the membership degrees. In contrast, Alpha [14] defines the similarity between two patterns as the smallest average membership degree among all slices of the merged pattern. Using that similarity, ALPHA hierarchically agglomerates patterns and selects agglomerates that are both "dense" and "anomalous." The difference between the similarity to the empty pattern $\varnothing$ and that to the parent pattern in the dendrogram formalizes the trade-off.

Several algorithms [15–20] approximately factorize n-way 0/1 tensors. As a consequence, the membership degrees in a fuzzy tensor must be rounded to 0/1 before applying any of those algorithm. The Boolean rank- r CANDECOMP/PARAFAC (CP) decomposition of a tensor $T \in \{0,1\}^{\prod_{i=1}^{n} D_i}$ aims to discover n matrices $A^1 \in \{0,1\}^{|D_1|\times r}, \cdots, A^n \in \{0,1\}^{|D_n|\times r}$ 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 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 set of n-tuples with value 1 is a pattern. The r patterns minimizing the objective function are usually large and dense, *i.e.*, they include many n-tuples whose average membership degree (the proportion of 1 in the sub-tensor) is high.

BCP_ALS [19] heuristically seeks a good CP decomposition of a 0/1 tensor by Alternating Least Squares. Distributed Boolean Tensor Factorization (DBTF) [20] distributes that method on the Spark framework and exhibits near-linear scalability w.r.t. the size of the 0/1 tensor, its density, the rank r (a parameter) of the factorization and the number of machines. Walk'n'Merge [16] discovers patterns through 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. Data-Peeler-like patterns (*i.e.*, all-ones sub-tensors) are added to the discovered patterns exceeding a minimal density threshold and pairs of patterns are merged if the result is sufficiently dense. Finally, the patterns that most decrease the objective function are output, a greedy process that stops when the model would overfit the data according to the Minimal Description Length principle.

Nonnegative tensor decomposition techniques can be applied to fuzzy tensors. Nevertheless, they model the data in a fundamentally different way: elements of the n dimensions are individually weighted, not patterns. The Cancer matrix (not tensor) factorization technique [21] is closest to this work: it substitutes the traditional addition for the max operator and can minimize the Frobenius norm.

Mirkin and Kramarenko map pattern mining in 0/1 tensors to a regression problem: [2] the set $\mathcal{X}$ of relevant patterns *explains* the tensor T. 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, under the convention $\max_{X\in\varnothing} \lambda_X = 0$, that T_t is

$$\hat{T}_t = \lambda_0 + \max_{X\in\mathcal{X} \text{ s.t. } t\in X} \lambda_X, \tag{1}$$

where the intercept $\lambda_0 \in [0,1]$, which can be seen as a similarity shift, is fixed by the analyst. A greater λ_0 favors the discovery of a model with more, smaller and denser patterns. Ordinary least squares guide the selection of the model, *i.e.*, the model aims to minimize the residual sum of squares:

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

Encoding $\mathcal{X}$ in the factors $A^1, \cdots, A^n$ of a rank-$|\mathcal{X}|$ Boolean CP decomposition, presented earlier, $RSS_T(\mathcal{X}) = \| T - \lambda_0 1 - \max_{k=1}^{|\mathcal{X}|} \lambda_k A^1_{:k} \otimes \cdots \otimes A^n_{:k} \|^2$. The Boolean CP decomposition therefore aims to minimize RSS_T with $\lambda_0 = 0$ and $\forall X \in \mathcal{X}, \lambda_X = 1$. In contrast, in the disjunctive box cluster model, every λ_X is estimated so that RSS_T is minimized.

The *TriclusterBox* algorithm repeatedly searches for one single pattern X that locally minimizes $X \mapsto RSS_T(\{X\})$. [2]. In that framework, λ_X must be $\frac{\sum_{t\in X}(T_t - \lambda_0)}{|X|}$ and $\lambda_X + \lambda_0$ is the density of X, a value that the analyst easily interprets. Minimizing $X \mapsto RSS_T(\{X\})$ equates to maximizing the function $g : X \mapsto |X|\lambda_X^2$, *i. e.*, after the similarity shift, the area times the squared density. *TriclusterBox* grows every pattern $\left(\prod_{i=1}^{n-1}\{t_i\}\right) \times \{t_n \in D_n \mid T_{t_1,\ldots,t_n} = 1\}$ obtained for all $(t_1, \ldots, t_{n-1}) \in \prod_{i=1}^{n-1} D_i$. Each of those $\prod_{i=1}^{n-1} |D_i|$ all-ones sub-tensors is the starting point of a hill-climbing (local) maximization of g. Every iteration leads to a pattern involving one more or one less element of any of the n dimensions. Once g cannot increase, the hill-climbing stops and the final pattern is added to $\mathcal{X}$, the model.

4. FITTING A DISJUNCTIVE BOX CLUSTER MODEL

Given a fuzzy tensor, the present proposal discovers patterns that heuristically minimize the residual sum of squares (2) of the disjunctive box cluster model (1). Three successive steps are taken:

1. Mine fragments of the patterns to discover;
2. Grow every fragment into a super-pattern that locally maximizes g in the space of the cardinalities of its dimensions;
3. Select a subset of the grown fragments so that Model (1) fits but does not overfit the data.

In the experimental section, multidupehack [11] (see Section 3.1) provides the so-called *fragments*: small patterns that are subpatterns of those in the desired disjunctive box cluster model. Another algorithm can be used though. The algorithmic contributions of this article deal with Steps 2 and 3. Sections 4.1 and 4.2 detail

two different ways (that can be successively applied) to grow a fragment (Step 2). Section 4.3 proposes a forward selection for Step 3.

4.1. Bigfoot

Like *TriclusterBox* [2], the algorithm proposed in this section repeatedly searches by hill-climbing for one pattern that locally minimizes the residual sum of squares (2) of the disjunctive box cluster model (1), *i.e.*, that locally maximizes $g : X \mapsto |X|\lambda_X^2$ with $\lambda_X = \frac{\sum_{t\in X}(T_t-\lambda_0)}{|X|}$ (see Ref. [2] for a proof of the equivalence). However, the *locality* is differently defined. *TriclusterBox* [2] considers two patterns $X = \prod_{i=1}^n X_i$ and $Y = \prod_{i=1}^n Y_i$ neighbors (*i.e.*, an iteration of the hill-climbing procedure 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 article, Y is a neighbor of X if and only if it includes the initial pattern and $|\cup_{i=1}^n Y_i| - |\cup_{i=1}^n X_i| = 1$. Unless X is the initial pattern, called *fragment* from now on, X therefore has more neighbors, the pattern space is more thoroughly explored and a higher local maximum of g is usually reached.

The fuzzy tensor T and the set of fragments to grow $\mathcal{F}$ are the inputs of Algorithm 1, named Bigfoot.[1] It outputs a set $\mathcal{C}$ of patterns that are candidate explanatory variables for the disjunctive box cluster model. Line 4 selects the fragment $F \in \mathcal{F}$ that best complements the previously discovered patterns, *i.e.*, that minimizes the function $G \mapsto RSS_T(\mathcal{C} \cup \{G\})$, and that is not included in any of them (line 18). At the first iteration, F is the fragment maximizing g. Later, that selection criterion penalizes fragments that largely overlap with the previously discovered patterns.

Computing $F = argmin_{G\in\mathcal{F}} RSS_T(\mathcal{C} \cup \{G\})$ is not as computationally expensive as it appears at first sight. At a given iteration of Algorithm 1, $RSS_T(\mathcal{C})$ is a constant value. As a consequence, $F = argmin_{G\in\mathcal{F}}(RSS_T(\mathcal{C} \cup \{G\}) - RSS_T(\mathcal{C}))$. For any pattern $G \in \mathcal{F}$, the models $\mathcal{C}$ and $\mathcal{C} \cup \{G\}$ predict the same membership degree for $t \notin G$, hence the same residual. That is why the difference $RSS_T(\mathcal{C} \cup \{G\}) - RSS_T(\mathcal{C})$ only depends on the values T_t and $\hat{T}_t$ for $t \in G$. That observation lowers the seemingly $O(|\mathcal{F}|\prod_{i=1}^n |D_i|)$ time complexity of line 4 to $O(\sum_{G\in\mathcal{F}} |G|)$. Moreover, the worst fragments in $\mathcal{F}$ need not even be considered. To not further disrupt the high-level presentation of Bigfoot, detailed explanations are in Appendix.

The fragment F is the starting point (line 5) of the hill-climbing optimization (lines 6–16). It searches a local maximum of g in the space of the numbers of elements $(\sigma_1, \dots, \sigma_n) \in \mathbb{N}^n$ that the super-patterns of F involve, in each of the n dimensions. Any of those numbers (line 9) can be incremented (line 10) during the search, which stops at line 16 if the pattern $X^{\max}$, reached at the previous iteration, is a local maximum of g. If so, line 17 adds $X^{\max}$ to the set of candidate explanatory variables for the disjunctive box cluster model. Otherwise (at least one neighbor of $X^{\max}$ evaluated g to a value exceeding $g(X^{\max})$), $X^{\max}$'s neighbor maximizing g became the best pattern so far (line 13) and the hill-climbing optimization goes on.

Algorithm 1's efficiency mainly depends on the time it takes to find the pattern $X = \prod_{i=1}^n X_i$ that maximizes g, among the super-patterns of F with $|X_1| = \sigma_1, \cdots, |X_n| = \sigma_n$. The function f, called at

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

Algorithm 1: `Bigfoot`

Input: fuzzy tensor T, intercept $\lambda_0 \in [0, 1]$, fragment set $\mathcal{F}$
Output: patterns in T that locally maximize g

1 $\mathcal{C} \leftarrow \emptyset$
2 $\mathcal{F}_\cup \leftarrow \cup_{F\in\mathcal{F}} F$
3 **while** $\mathcal{F} \neq \emptyset$ **do**
4 $\quad F \leftarrow \arg\min_{G\in\mathcal{F}} RSS_T(\mathcal{C} \cup \{G\})$
5 $\quad X^{\max} \leftarrow F$
6 $\quad$ **repeat**
7 $\quad\quad$ *optimal* $\leftarrow$ **true**
8 $\quad\quad (\sigma_1, \dots, \sigma_n) \leftarrow (|X_1^{\max}|, \dots, |X_n^{\max}|)$
9 $\quad\quad$ **for** $i = 1 \to n$ **do**
10 $\quad\quad\quad \sigma_i \leftarrow \sigma_i + 1$
11 $\quad\quad\quad X \leftarrow f(T, \mathcal{F}_\cup, F, \sigma_1, \dots, \sigma_n)$
12 $\quad\quad\quad$ **if** $g(X) > g(X^{\max})$ **then**
13 $\quad\quad\quad\quad X^{\max} \leftarrow X$
14 $\quad\quad\quad\quad$ *optimal* $\leftarrow$ **false**
15 $\quad\quad\quad \sigma_i \leftarrow \sigma_i - 1$
16 $\quad$ **until** *optimal*
17 $\quad \mathcal{C} \leftarrow \mathcal{C} \cup \{X^{\max}\}$
18 $\quad \mathcal{F} \leftarrow \{F \in \mathcal{F} \mid F \not\subseteq X^{\max}\}$
19 **return** $\mathcal{C}$

line 11, computes X by ILP. The problem is the maximization of

$$\sum_{t\in\mathcal{F}_\cup} w_t(T_t - \lambda_0) \tag{3}$$

under the following constraints:

$$\forall t \in \mathcal{F}_\cup, 0 \leqslant \sum_{i=1}^n x_{t_i} - nw_t \leqslant n - 1 \tag{4}$$

$$\forall t \in F, \forall i \in \{1, \dots, n\}, x_{t_i} = 1 \tag{5}$$

$$\forall i \in \{1, \dots, n\}, \sum_{e\in\{t_i \mid t\in\mathcal{F}_\cup\}} x_e = \sigma_i \tag{6}$$

$$\forall t \in \mathcal{F}_\cup, w_t \in \{0, 1\} \tag{7}$$

$$\forall e \in \cup_{i=1}^n \{t_i \mid t \in \mathcal{F}_\cup\}, x_e \in \{0, 1\} \tag{8}$$

Constraints (7) and (8) force all variables w_t and x_e to be either 0 or 1. $w_t = 1$ if the n-tuple 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 $X = \prod_{i=1}^n \{e \in D_i \mid x_e = 1\}$. As detailed later, Constraint (4) forces the values of all variables

w_t and x_e to be coherent with the syntactic definition of a pattern (see Section 2). Constraint (5) forces the returned pattern X to be a super-pattern of the grown fragment F, *i. e.*, $X \supseteq F$. Finally, Constraint (6) forces $X = \prod_{i=1}^{n} X_i$ to satisfy $\forall i \in \{1, \dots, n\}, |X_i| = \sigma_i$. Given those constant cardinalities, $|X|$ is constant too. It is $\prod_{i=1}^{n} \sigma_i$. That is why maximizing $g(X) = |X|\lambda_X^2 = \frac{\left(\sum_{t \in X}(T_t - \lambda_0)\right)^2}{|X|}$ amounts to maximizing $\sum_{t \in X}(T_t - \lambda_0)$.

However, according to Problem (3), f maximizes a slightly different sum: $\sum_{t \in \mathcal{F}_\cup} w_t(T_t - \lambda_0)$. Given $\mathcal{F}_\cup$'s definition, at line 2 of Algorithm 1, the terms that relate to n-tuples that are not in any fragment are missing. f therefore assumes $\forall t \notin \mathcal{F}_\cup, T_t = \lambda_0$. In other terms, f maximizes g in an approximation $\tilde{T}$ of the tensor T: $\tilde{T}_t = \begin{cases} T_t \text{ if } t \in \mathcal{F}_\cup \\ \lambda_0 \text{ otherwise} \end{cases}$. If the fragments include all the n-tuples in the patterns composing the desired model $\mathcal{X}$, *i. e.*, if $\cup_{X \in \mathcal{X}} X \subseteq \mathcal{F}_\cup$, then the approximation is harmless. However, the fragments do not *need* to include all these n-tuples. Indeed, 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 absent from $\mathcal{F}_\cup$.

The reason why f assumes $\forall t \notin \mathcal{F}_\cup, T_t = \lambda_0$ is simple: for a fast resolution of the ILP problem, its number of variables ought to be small. Thanks to the assumption, Constraints (7) and (8) "only" define $|\mathcal{F}_\cup|$ variables w_t and $\sum_{i=1}^{n} |\{t_i \mid t \in \mathcal{F}_\cup\}|$ variables x_e 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 because the ILP solver only branches on the x_e variables. Indeed, a valuation of the x_e variables implies a unique valuation of the (more numerous) w_t variables, according to Constraint (4), which equates to

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

Finally, at line 11 of Algorithm 1, giving $g(X^{\max})$ as an additional argument to the function f allows to add the following constraint for a faster resolution of every ILP problem, without any alteration of Algorithm 1's output:

$$\sum_{t \in \mathcal{F}_\cup} w_t(T_t - \lambda_0) > \sqrt{g(X^{\max}) \prod_{i=1}^{n} \sigma_i}. \tag{9}$$

$\prod_{i=1}^{n} \sigma_i$ is the area of the pattern X that f must return. Under the assumption $\forall t \notin \mathcal{F}_\cup, T_t = \lambda_0$, Constraint (9) equates to $g(X) > g(X^{\max})$. Since f's output only matters if the test at line 12 of Algorithm 1 passes, Constraint (9) safely prunes the search space.

4.2. Bigfoot-LR

f, called at line 11 of Algorithm 1, may take exponential time to solve the ILP problem. For a polynomial time complexity, f can, instead, solve the linear relaxation of the problem, where Constraint (7) is substituted by

$$\forall t \in \mathcal{F}_\cup, w_t \in [0, 1].$$

To maximize $\sum_{t \in \mathcal{F}_\cup} w_t(T_t - \lambda_0)$ (Problem (3)), w_t must be maximal for all $t \in \mathcal{F}_\cup$. However, Constraint (4) (the only constraint involving the w_t variables) forces $0 \leqslant \sum_{i=1}^{n} x_{t_i} - n w_t$. As a consequence, every w_t must be valuated to $\frac{\sum_{i=1}^{n} x_{t_i}}{n}$ and the linearly relaxed problem is the maximization of

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

under Constraints (5), (6) and (8).

Taking the n-tuples in $\mathcal{F}_\cup$ slice by slice (definition in Section 2), the problem becomes the maximization of

$$\frac{1}{n} \sum_{i=1}^{n} \sum_{e \in D_i} \left(x_e \sum_{t \in \mathcal{F}_\cup \text{ s.t. } t_i = e} (T_t - \lambda_0) \right) \tag{10}$$

under Constraints (5), (6) and (8).

Given the inputs of Algorithm 1, the innermost sums (one sum for each element $e \in \cup_{i=1}^{n} D_i$) are constants. They only need to be computed once, before running Algorithm 1. A greedy algorithm solves the problem in an exact way. First of all, Constraint (5) forces $x_e = 1$ for all elements e involved in the fragment F. Then, additional x_e variables are set to 1 to satisfy Constraint (6): those with e in the proper dimension and the greatest $\sum_{t \in \mathcal{F}_\cup \text{ s.t. } t_i = e} (T_t - \lambda_0)$. Lines 2 and 3 in Algorithm 2 respectively select those two types of elements.

Algorithm 2: f solving Problem (10)

Input: every dimension $(D_1, \dots, D_n)$ increasingly ordered by $\sum_{t \in \mathcal{F}_\cup \text{ s.t. } t_i = e} (T_t - \lambda_0)$, fragment F, numbers of elements $\sigma_1, \dots, \sigma_n$

Output: solution to Problem (10)

1 **for** $i \leftarrow 1$ ***to*** n **do**
2 $\quad X_i \leftarrow \{t_i \mid t \in F\}$
3 $\quad X_i \leftarrow X_i \cup \{\text{last } \sigma_i - |X_i| \text{ elements in } D_i \setminus X_i\}$
4 **return** $\prod_{i=1}^{n} X_i$

Bigfoot-LR is the name given to Algorithm 1 when f solves Problem (10) with Algorithm 2. Algorithm 2 is fast. It runs in $O(\sum_{i=1}^{n} \sigma_i)$ time. However, only growing the fragments with n-tuples in the densest slices of $\mathcal{F}_\cup$ is naive. Section 5 shows that Bigfoot-LR returns patterns that are only slightly larger than the fragments.

Nevertheless, Bigfoot can further grow those patterns. They become the argument $\mathcal{F}$ of a second execution of Algorithm 1, this time without the linear relaxation of the ILP problem. Being larger than the initial fragments, Bigfoot-LR's patterns save some iterations of Bigfoot's hill-climbing. On the other hand, altogether, Bigfoot-LR's patterns include more n-tuples than the initial fragments and solving the individual ILP problems takes more time. *TriclusterBox*, discussed in Section 3.2, is less naive than Bigfoot-LR. However, it cannot provide "larger fragments" to Bigfoot. Indeed, Bigfoot would not grow them because they already locally maximize g.

4.3. Forward Selection

The disjunctive box cluster model composed of *all* the patterns that Bigfoot, Bigfoot-LR or Bigfoot-LR+Bigfoot outputs overfits the fuzzy tensor. A well-chosen subset of those patterns usually makes a statistically more relevant model, *i.e.*, a simpler (with fewer patterns) disjunctive box cluster model (1) whose residual sum of squares (2) is not significantly higher. Stepwise regression aims to heuristically select such a subset of the candidate explanatory variables (the computed patterns). The search of a trade-off between the goodness of fit and the simplicity is formalized as the minimization of a measure, *e.g.*, the *Akaike information criterion* (*AIC*) [3]. Assuming that, for any subset $\mathcal{X}$ of the set $\mathcal{C}$ of computed patterns, the residuals $T_t - \hat{T}_t$ follow an independent identical normal distribution with zero mean, $AIC_T(\mathcal{X})$ is

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

That formula considers that the models, the subsets of $\mathcal{C}$, only predict the membership degrees of the n-tuples in $\cup_{C\in\mathcal{C}} C$. It disregards every n-tuple $t \notin \cup_{C\in\mathcal{C}} C$ because $\forall\mathcal{X} \subseteq \mathcal{C}$, $\hat{T}_t = \lambda_0$. Besides, unless λ_0 is the average membership degree over these n-tuples, the mean of the residuals would not be zero, a violation of one of the assumptions behind the formula. Those same arguments stand when it comes to assessing the quality of the selected model $\mathcal{X} \subseteq \mathcal{C}$, *i.e.*, AIC_T should disregard every n-tuple $t \notin \cup_{X\in\mathcal{X}} X$. That is why the proposed stepwise regression is recursive: it first selects a subset $\mathcal{X}_1$ of $\mathcal{C}$, then a subset $\mathcal{X}_2$ of $\mathcal{X}_1$ (substituting $\mathcal{C}$ by $\mathcal{X}_1$ in the formula above), etc. That process stops at iteration k if $\mathcal{X}_k = \mathcal{X}_{k-1}$ (a fixed point) and $\mathcal{X}_k$ is the disjunctive box cluster model that is returned.

Algorithm 3 formalizes the recursive forward selection (the chosen stepwise regression technique) of the disjunctive box cluster model $\mathcal{X} \subseteq \mathcal{C}$. At every iteration, line 3 takes from $\mathcal{C}$ the pattern that, added to $\mathcal{X}$, minimizes RSS_T (and AIC_T), *i.e.*, that best complements the current model. That selection is analogous to that of line 4 in Algorithm 1. As explained in Section 4.1, it takes $O(\sum_{C\in\mathcal{C}} |C|)$-time and favors the selection of a pattern not intersecting much the previously selected patterns, because the predicted membership degrees for the n-tuples in the intersection can only marginally increase. If adding to $\mathcal{X}$ any more pattern, taken in $\mathcal{C}$, 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 3: `forward-select`

Input: fuzzy tensor T, intercept $\lambda_0 \in [0,1]$, candidate pattern set $\mathscr{C}$
Output: $\mathscr{X} \subseteq \mathscr{C}$ fitting T through Model (1)

1 $\mathscr{X} \leftarrow \emptyset$
2 **while** $\mathscr{X} \neq \mathscr{C}$ **do**
3 $X \leftarrow \arg\min_{C\in\mathscr{C}} RSS_T(\mathscr{X} \cup \{C\})$
4 **if** $AIC_T(\mathscr{X} \cup \{X\}) > AIC_T(\mathscr{X})$ **then**
5 **return** `forward-select`$(T, \lambda_0, \mathscr{X})$
6 $\mathscr{X} \leftarrow \mathscr{X} \cup \{X\}$
7 **return** $\mathscr{X}$

5. EXPERIMENTAL VALIDATION

multidupehack [11], Bigfoot, Bigfoot-LR and *TriclusterBox* [2] are all distributed under the terms of the GNU GPLv3. They are implemented in C++ and compiled by GCC 5.4.1 with the O2 optimizations. The implementation in C of Walk'n'Merge [16] was provided by its authors. DBTF [20] is only distributed in binary form. All experiments are performed on a GNU/Linux™ system. It runs on top of 2.4 GHz cores, 12 MB of cache and 32 GB of RAM. Bigfoot uses IBM ILOG CPLEX Optimizer [22] v12.6.0 to solve the ILP problems. https://gitlab.com/lucasmaciel82/Bigfoot hosts the datasets, the source codes (including for the generation of synthetic tensors) and the scripts to run all the experiments.

5.1. Real-world Tensors

5.1.1. A 3-way tensor

The first real-world fuzzy tensor used in this section 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 13th to April 6th). The influence (*i.e.*, the membership degree) is here defined from how many times the messages were "retweeted" and from the overall popularity of the team: the numbers of retweets for a given team are multiplied by a constant chosen so that the sum of the normalized numbers is the average number of retweets per team; a logistic function, with a growth rate of 0.5 and centered on 10 normalized retweets (as shown in Figure 1), turns each normalized number of retweets into an influence, in $[0,1]$. Summing all influences gives 40, 167.3, *i.e.*, $\frac{40{,}167.3}{170{,}670\times29\times12} \approx 0.0006$ times the maximal possible value. Because $\lambda_0 = 0.0006$ minimizes $RSS_T(\emptyset)$, it can be considered the default setting. Here, λ_0 is set to 0.1, to discover more and denser (although smaller) patterns.

multidupehack [11] provides the fragments. Three minimal size constraints are enforced (see Section 3.1): at least eight users, three teams and four weeks. With the upper-bounds $\epsilon_{\text{user}} = 3$, $\epsilon_{\text{team}} = 8$ and $\epsilon_{\text{week}} = 6$ (again, see Section 3.1), there are 1,183,653 fragments. multidupehack returns them within 47.4 seconds. Given

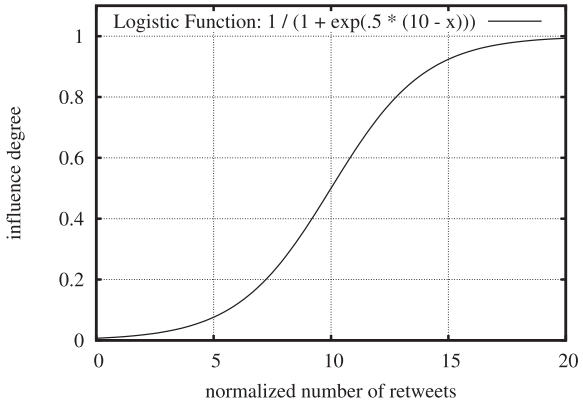

Figure 1 | Curve of the logistic function chosen to turn normalized numbers of retweets into influence degrees: 10 normalized retweets map to "moderately influential."

their minimal sizes and the upper-bounds, any slice of any fragment has, at least, a 0.75 density. Bigfoot-LR takes 2 minutes and 20 seconds to turn the 1,183,653 fragments into only 44 distinct patterns. The high degree of overlap between multidupehack's patterns and Bigfoot-LR's addition of elements by decreasing order of density together explain why that number is so small: many fragments grow into a same pattern. Bigfoot processes Bigfoot-LR's 44 patterns within 52 seconds and the forward selection only keeps two patterns. The first pattern is large and rather dense: $\lambda_X + \lambda_0 = 0.81$. It involves all 12 weeks, 110 influential supporters and journalists and 13 famous teams (most of them from the Southeast region of Brazil), hence 17,160 3-tuples. The second pattern is smaller, 150 3-tuples, but denser, $\lambda_X + \lambda_0 = 0.85$. It only involves teams and supporters from the South region of Brazil. Bigfoot alone takes more than ten hours (at which point it was aborted) to process the 1,183,653 fragments.

However, Bigfoot can directly process, in a reasonable time, an alternative collection of fragments. By reducing the minimal number of teams in a fragment to two but forcing these fragments to be denser (with $\epsilon_{\text{user}} = 0.5$, $\epsilon_{\text{team}} = 2$ and $\epsilon_{\text{week}} = 1$, *i.e.*, a minimal possible density of 0.875), multidupehack takes 1.1 second to provide 359 fragments. Within 6 hours and 54 minutes, Bigfoot and the forward selection turn those fragments into six patterns. The first three columns of Table 3 list them in the order Algorithm 3 selects them, *i.e.*, from the pattern that most reduces RSS_T to the one that least contributes to the model. Their first two dimensions are large. That is why Table 3 only reports their cardinalities.

Every discovered pattern only involves teams from one single state. For example, the first pattern involves four teams from Rio de Janeiro. 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 identifiable users involved in a given pattern are either journalists, who are indeed influential, or supporters of one of the teams in the pattern. The first four patterns do not intersect, a property favored by the forward selection, as explained in the last paragraph of Section 4.3. The last two patterns 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 last column of Table 3 gives the densities of the patterns. The patterns are large and dense. Within 2.7 seconds, Bigfoot-LR grows the same 359 fragments into 46 slightly larger fragments.

TriclusterBox [2], Walk'n'Merge [16] and DBTF [20] only handle 0/1 tensors. To use them, every membership degree in the fuzzy tensor is rounded to 0 or 1. It takes *TriclusterBox* ten hours of computation to only grow one single pattern out of $12 \times 29 = 348$ (see Section 3.2). Despite many attempts with different minimal size and density parameters, Walk'n'Merge only outputs two patterns within 2 hours and 19 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 to discover. However, even if that parameter is set to 30, DBTF does not discover any pattern: it either crashes or one of the returned factors is null.

Table 3 Disjunctive box cluster model, discovered by Bigfoot, of the $170,670 \times 12 \times 29$ retweet tensor.

$\lvert X_{\text{user}}\rvert$	$\lvert X_{\text{weeks}}\rvert$	X_{teams}	$\lambda_X + \lambda_0$
31	12	{Botafogo, Flamengo, Fluminense, Vasco}	0.702
28	11	{Corinthians, Palmeiras, Santos}	0.754
14	10	{Avaí, Figueirense}	0.864
17	7	{Grêmio, Internacional}	0.859
15	11	{Flamengo, Fluminense}	0.792
17	12	{Flamengo, Vasco}	0.801

5.1.2. A 4-way tensor

The second real-world application analyzes 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 $7 \times 24 \times 327 \times 327$ fuzzy tensor has a 0.005 density. λ_0 is set to that value.

To discover in the 7×24 graphs some kind of cross-graph quasi-cliques, *i.e.*, 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 do so [11]. To have Bigfoot explore that same restricted pattern space, one additional constraint is added to the ILP problem that *f* solves at line 11 of Algorithm 1:

$$\forall t \in \mathcal{T}_{\cup}, x_{t_{\text{departure}}} = x_{t_{\text{arrival}}}.$$

The hill-climbing procedure has to be modified too: at line 9 of Algorithm 1, one single index stands for both the departure and the arrival stations and the related iterations increment/decrement both $\sigma_{\text{departure}}$ and σ_{arrival} at lines 10 and 15. Those simple changes, which demonstrate the adaptability of the proposal, reduce the time requirements because more constrained ILP problems are easier and because hill-climbing in a smaller pattern space is faster.

With $\epsilon_{\text{departure}} = \epsilon_{\text{arrival}} = 21.6$, $\epsilon_{\text{hour}} = 14.4$ and $\epsilon_{\text{day}} = 28.8$, multidupehack takes 2 minutes and 44 seconds to return 5,462 fragments with at least four stations, six one-hour periods and three days (hence a minimal possible density of 0.7). Bigfoot grows them within 5 hours and 40 minutes. Algorithm 3 then selects two patterns. Figures 2 and 3 show the geographic positions of the stations involved in those patterns. Their captions report the associated days of the week and periods of the day. The available implementations of *TriclusterBox*, Walk'n'Merge and DBTF are restricted to mining 3-way 0/1 tensors: they cannot be used here.

5.2. Synthetic Tensors

The actual patterns to discover in real-world tensors are unknown. To assess to what extent 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 3-tuples with membership degrees equal to 1) of sizes $6 \times 6 \times 6$ are randomly

planted in a null $32 \times 32 \times 32$ tensor. With those settings, the patterns often overlap, what happens in real-world contexts too, *e.g.*, in Table 3 or in Figures 2 and 3.

Considering every 0 or 1 in a "perfect tensor" as the output of a Bernoulli variable with parameter p, the probability of a 1, *inverse transform sampling* allows to noise the tensor. The posterior distribution of p is the beta distribution of parameters α and β, which 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. 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. More correct observations lead to membership degrees that are closer to the values in the "perfect" tensor. Figure 4 plots the inverses of cumulative beta

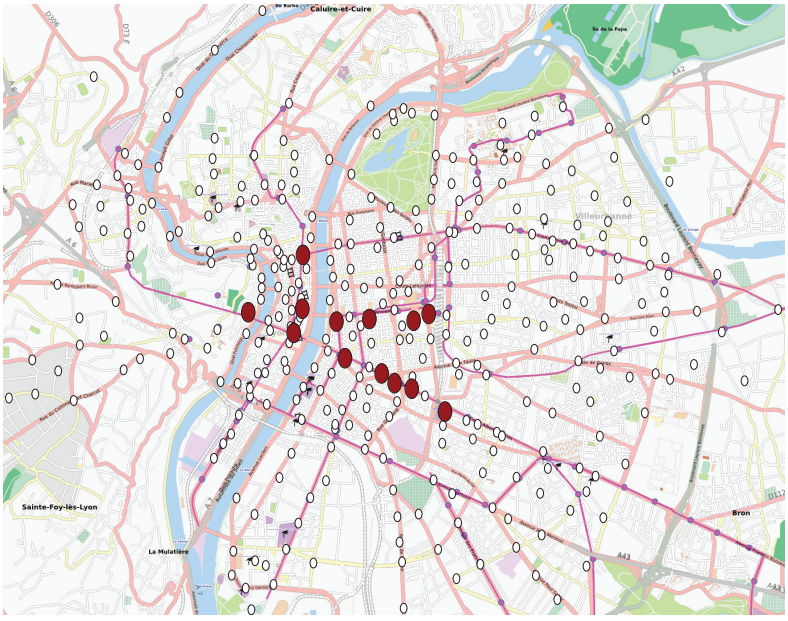

Figure 2 | Thirteen large stations in the working and commercial districts of Lyon. They exchange many bicycles everyday from midday to 8pm, except on Sundays, when the shops are closed. $|X| = 8.112$ and $\lambda_X + \lambda_0 = 0.383$.

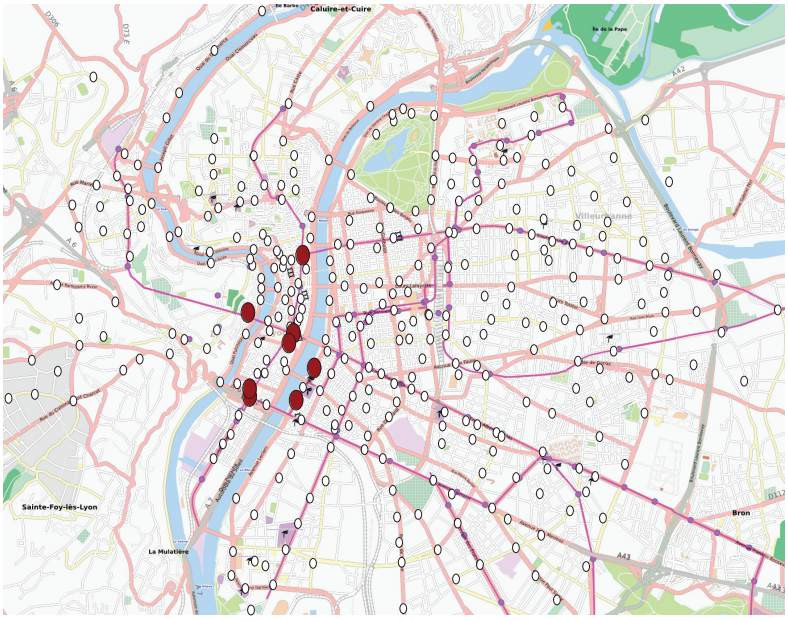

Figure 3 | Eight large stations around the main squares of Lyon, in its historical center. Everyday, from 8am to 10am and from midday to 9pm, many users ride between those stations. $|X| = 4,928$ and $\lambda_X + \lambda_0 = 0.256$.

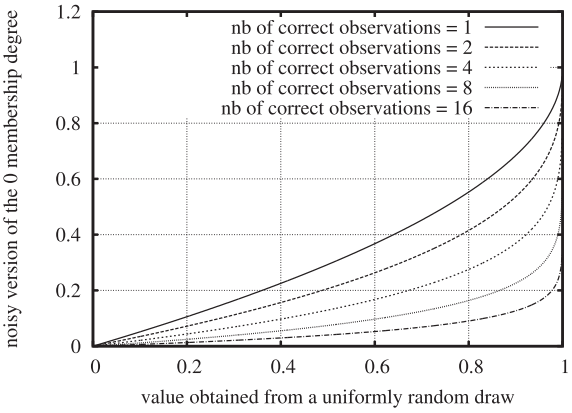

Figure 4 Inverses of cumulative beta distributions used to noise a membership degree at 0 in a "perfect" tensor. More "correct observations" mean less noise.

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 observation 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 $\frac{|P \cap X|}{|P \cup X|}$ measures their similarity. Given $\mathcal{P}$, the planted patterns, and $\mathcal{X}$, the patterns discovered in the related fuzzy tensor, the *quality* of $\mathcal{X}$ is here defined as

$$\frac{\left|\bigcup_{P \in \mathcal{P}} \left(P \cap argmax_{X \in \mathcal{X}} \frac{|P \cap X|}{|P \cup X|}\right)\right|}{\left|\bigcup_{P \in \mathcal{P}} P \cup \bigcup_{X \in \mathcal{X}} X\right|}.$$

That 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 absence 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 penalize the discovery of supernumerary overlapping patterns. That is why the following experimental results report as well the number of patterns that are discovered, $|\mathcal{X}|$.

Figure 5 shows the quality of the discovered patterns, their numbers, and the run times, in function of the level of noise in the 3-way fuzzy tensor. In this section, λ_0 is always set to 0. For a given level of noise, all results are averages over eight randomly generated tensors. multidupehack [11] provides the fragments, with at least three elements of every dimension. Different upper-bounds $(\epsilon_1, \epsilon_2, \epsilon_3)$ are tested so that the fragments can be more or less dense. The upper-bound is always the same whatever the dimension, *i.e.*, $\epsilon_1 = \epsilon_2 = \epsilon_3 = \epsilon$. It relates to μ, whose values are written above the plots: $\epsilon = 3^2(1-\mu)$. For instance, for $\mu = 0.6$ (rightmost plots), multidupehack's upper-bounds are (3.6, 3.6, 3.6). Given multidupehack's definition of a fragment and the minimal size constraints, $\mu = 1 - \frac{\epsilon}{3^2}$ is the minimal possible density for any slice of a fragment. Forward selection (see Section 4.3) is used to simplify the output of all methods but multidupehack's, so that the overall gains can be observed. Whatever the method, the forward selection improves the quality and, of course, decreases the number of patterns. The reported run times include that step.

multidupehack returns up to hundreds of thousands of fragments. They poorly match the planted patterns, unless the level of noise is very low (16 correct observations). Nevertheless, 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. Bigfoot is always the best performer. That is, when it is actually given fragments to grow and when it runs within one hour, the chosen timeout. Bigfoot can indeed require much time, especially to grow fragments of low density in noisy tensors. Bigfoot-LR is very fast but it returns terrible patterns, enhanced fragments that need more growing. Using Bigfoot to further grow them, Bigfoot-LR+Bigfoot often outputs the same patterns as Bigfoot alone. When it does not, the obtained qualities are slightly lower. Given the least dense fragments ($\mu = 0.6$), Bigfoot-LR+Bigfoot does not even take one tenth of the time Bigfoot requires. However, it may be slower, *e.g.*, to grow the fragments with $\mu = 0.7$ in the noisiest tensors. The end of Section 4.2 explains why. In presence of little noise (at least 4 correct observations), Bigfoot always recovers the four planted patterns, whatever the tested minimal density μ of the provided fragments. With $\mu = 0.8$, multidupehack returns too few fragments in the noisier tensors to recover the planted patterns, whereas $\mu = 0.7$ allows Bigfoot's patterns to exceed the 0.9 quality, even in the noisiest setting (1 correct observation).

Bigfoot and its variations handle n-way fuzzy tensors. In contrast, the state of the art only deals with 3-way 0/1 tensors. After rounding to 0 or 1 the membership degrees of the *same* noisy tensors used above, Bigfoot and Bigfoot-LR+Bigfoot can be compared to *TriclusterBox*, Walk' n'Merge and DBTF. The minimal density parameter of Walk'n' Merge's is set to 0.7; its other parameters to their default values. Walk'n'Merge uses the Minimal Description Principle to not overfit the tensor. *TriclusterBox* includes no such mechanism and the forward selection, described in Section 4.3, always improves its output. *TriclusterBox*'s results in Figure 6 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 sometimes returns fewer patterns. multidupehack still provides the fragments that Bigfoot and Bigfoot-LR+Bigfoot grow. It is configured as earlier and μ is set to 0.7. Here, that means any slice of any fragment contains at most two 3-tuples with null membership degrees.

Figure 6 shows that Bigfoot does not always reach the qualities reported in Figure 5: rounding to 0/1 harms the ability to recover the planted patterns. Despite that, 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. The competitors are faster though, especially in the noisiest settings.

6. 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.

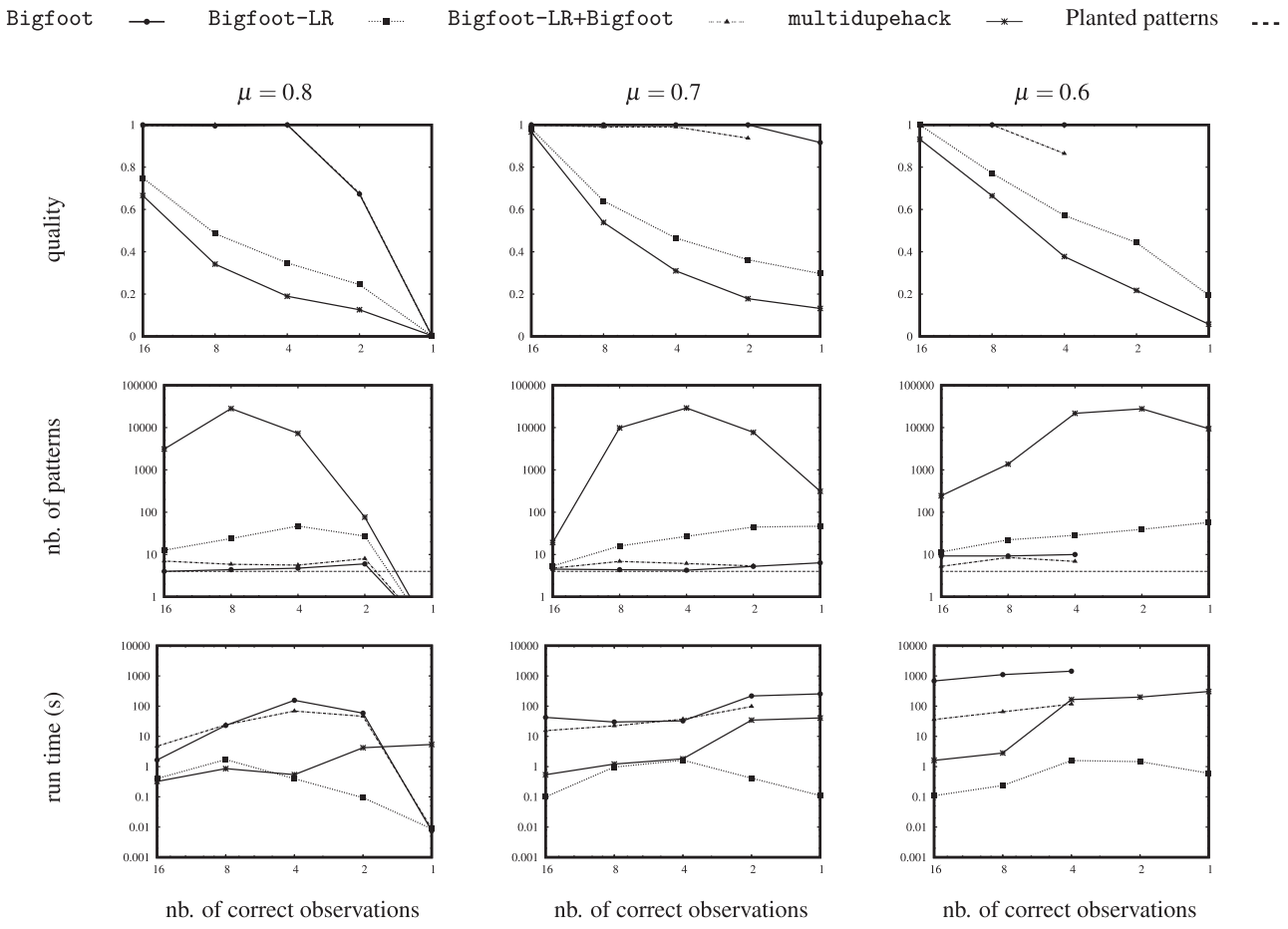

Figure 5 | Qualities, numbers of patterns and run times of multidupehack, Bigfoot, Bigfoot-LR and Bigfoot-LR+Bigfoot mining noisy 3-way tensors (the level of noise increases from left to right).

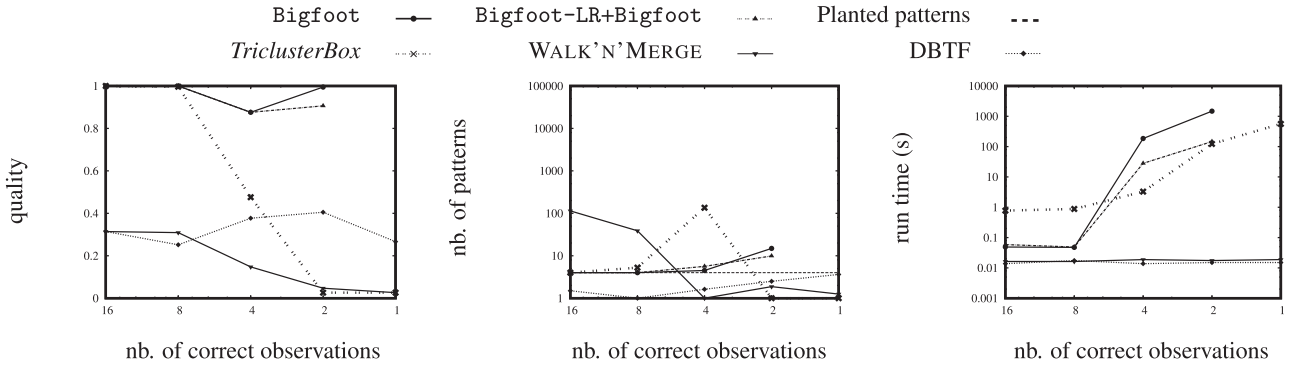

Figure 6 | Qualities, numbers of patterns and run times of the competing methods mining noisy 3-way 0/1 tensors (the level of noise increases from left to right).

Such a disjunctive box cluster model therefore is more informative than a Boolean CP factorization but it remains easy to interpret. Moreover, it suits fuzzy tensors and not only 0/1 tensors.

This article has presented the first solution to decompose fuzzy tensors into patterns. It grows pattern fragments and selects a small number of grown patterns to fit but not overfit the fuzzy tensor. Various techniques have been combined. In particular, every pattern is grown by hill-climbing and an integer linear model has been defined to have an existing solver efficiently find the pattern at the next iteration. Since solving ILP problems may take exponential time, a linearly relaxed variation of the problem has been presented as well. The related algorithm is less effective but can be used in a preprocessing step. Finally, a recursive forward selection composes the returned model, a subset of the grown patterns, by greedily minimizing the AIC.

Experiments have shown that the method successfully recovers patterns in noisy synthetic tensors. It even outperforms state-of-the-art approaches when the tensor is 0/1, a special case. Relevant patterns

have been found in two real-world fuzzy tensors with tens of millions of values. Future work includes extending the integer linear model and exploring other algorithmic approaches to even more accurately and efficiently fit a disjunctive box cluster model to a fuzzy tensor.

CONFLICTS OF INTEREST

The authors have no conflict of interest to declare.

AUTHORS' CONTRIBUTIONS

Lucas Maciel designed, implemented and evaluated most of the proposal. Jônatas Alves helped him with the experiments on synthetic tensors; Vinicius Fernandes dos Santos with the definition of the ILP problem; Loïc Cerf with the forward selection, the computation of $\operatorname{argmin}_{C \in \mathcal{C}} RSS_T(\mathcal{X} \cup C)$ and the generation of synthetic tensors. Together, Lucas Maciel and Loïc Cerf did most of the literature survey and most of the writing.

ACKNOWLEDGMENTS

The work has been partially funded by the *Fundação de Amparo à Pesquisa do Estado de Minas Gerais* under Grant number APQ-04224-16.

REFERENCES

[1] M.G. Thomason, Convergence of powers of a fuzzy matrix, J. Math. Anal. Appl. 57 (1977), 476–480.
[2] B.G. Mirkin, A.V. Kramarenko, Approximate bicluster and tricluster boxes in the analysis of binary data, in: S.O. Kuznetsov, D. Ślęzak, D.H. Hepting, B.G. Mirkin (Eds.), Proceeding of the 13th International Conference on Rough Sets, Fuzzy Sets, Data Mining and Granular Computing, Springer, Berlin, Heidelberg, Germany, 2011, pp. 248–256.
[3] H. Akaike, A new look at the statistical model identification, Trans. Automat. Control. 19 (1974), 716–723.
[4] H. Cheng, P.S. Yu, J. Han, AC-close: efficiently mining approximate closed itemsets by core pattern recovery, in Proceeding of the 6th IEEE International Conference on Data Mining, IEEE Computer Society, Hong Kong, China, 2006, pp. 839–844.
[5] A.K. Poernomo, V. Gopalkrishnan, Towards efficient mining of proportional fault-tolerant frequent itemsets, in Proceeding of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM Press, Paris, France, 2009, pp. 697–706.
[6] A.K. Poernomo, V. Gopalkrishnan, Mining statistical information of frequent fault-tolerant patterns in transactional databases, in Proceeding of the 7th IEEE International Conference on Data Mining, IEEE Computer Society, Omaha, NE, USA, 2007, pp. 272–281.
[7] L. Cerf, J. Besson, C. Robardet, J.-F. Boulicaut, Closed patterns meet n-ary relations, ACM Trans. Knowl. Discov. Data. 3 (2009), 1–36.
[8] D.I. Ignatov, D.V. Gnatyshak, S.O. Kuznetsov, B.G. Mirkin, Triadic formal concept analysis and triclustering: searching for optimal patterns, Mach. Learn. 101 (2015), 271–302.
[9] E. Spyropoulou, T. De Bie, M. Boley, Interesting pattern mining in multi-relational data, Data Mining Knowl. Discov. 28 (2014), 808–849.
[10] E. Spyropoulou, T. De Bie, Mining approximate multi-relational patterns, in Proceeding of the 2014 IEEE International Conf. on Data Science and Advanced Analytics, IEEE Computer Society, Shanghai, China, 2014, pp. 477–483.
[11] L. Cerf, W. Meira Jr., Complete discovery of high-quality patterns in large numerical tensors, in Proceeding of the 30th IEEE International Conference on Data Engineering, IEEE Computer Society, Chicago, IL, USA, 2014, pp. 448–459.
[12] E. Georgii, K. Tsuda, B. Schölkopf, Multi-way set enumeration in weight tensors, Mach. Learn. 82 (2011), 123–155.
[13] L. Zhao, M.J. Zaki, An effective algorithm for mining coherent clusters in 3D microarray data, in Proceeding of the 2005 ACM SIGMOD International Conference on Management of Data, ACM Press, Baltimore, MD, USA, 2005, pp. 694–705.
[14] L. Cerf, P.-N. Mougel, J.-F. Boulicaut, Agglomerating local patterns hierarchically with ALPHA, in Proceeding of the 18th ACM Conference on Information and Knowledge Management, ACM Press, Hong Kong, China, 2009, pp. 1753–1756.
[15] R. Bělohlávek, C. Glodeanu, V. Vychodil, Optimal factorization of three-way binary data using triadic concepts, Order. 30 (2013), 437–454.
[16] D. Erdős, P. Miettinen, Walk'n'Merge: a scalable algorithm for boolean tensor factorization, in Proceeding of the 13th IEEE International Conference on Data Mining, IEEE Computer Society, Dallas, TX, USA, 2013, pp. 1037–1042.
[17] I. Leenen, I. Van Mechelen, P. De Boeck, S. Rosenberg, INDCLAS: athree-way hierarchical classes model, Psychometrika. 64 (1999), 9–24.
[18] S. Metzler, P. Miettinen, Clustering boolean tensors, Data Mining Knowl. Discov. 29 (2015), 1343–1373.
[19] P. Miettinen, Boolean tensor factorizations, in Proceeding of the 11th IEEE International Conference on Data Mining, IEEE Computer Society, Vancouver, Canada, 2011, pp. 447–456.
[20] N. Park, S. Oh, U. Kang, Fast and scalable distributed boolean tensor factorization, in Proceeding of the 33rd IEEE International Conference on Data Engineering, IEEE Computer Society, San Diego, CA, USA, 2017, pp. 1071–1082.
[21] S. Karaev, P. Miettinen, Cancer: another algorithm for subtropical matrix factorization, in: P. Frasconi, N. Landwehr, G. Manco, J. Vreeken (Eds.), Proceeding of the European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, Cham, Switzerland, 2016, pp. 576–592.
[22] E. Robert, Bixby, ILOG, and IBM, CPLEX Optimizer, 1988. https://www.ibm.com/analytics/cplex-optimizer/

APPENDIX

This appendix deals with the efficient computation of $argmin_{C \in \mathcal{C}} RSS_T(\mathcal{X} \cup \{C\})$, at line 4 of Algorithm 1 and at line 3 of Algorithm 3. Section 4.1 has already shown its time complexity is $O(\sum_{C \in \mathcal{C}} |C|)$, because, at a given iteration of Algorithm 1 or 3, $RSS_T(\mathcal{X})$ is a constant and the difference $RSS_T(\mathcal{X} \cup \{C\}) - RSS_T(\mathcal{X})$ only depends on the membership degrees and the predicted membership degrees of the n-tuples in $C \in \mathcal{C}$.

However, the worst patterns in $\mathcal{C}$ need not even be considered if information computed in previous iterations of those algorithms is kept in memory. The idea is to store along every pattern $C \in \mathcal{C}$, a lower bound of $RSS_T(\mathcal{X}' \cup \{C\}) - RSS_T(\mathcal{X}')$, where $\mathcal{X}'$ 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 $C \in \mathcal{C}$ exceeds the smallest difference of RSS_T found so far, C cannot be the pattern to select. Given (1) and (2), if $\hat{T}'_t$ is the membership degree that the future model $\mathcal{X}'$ predicts for the n-tuple $t \in C$, $RSS_T(\mathcal{X}' \cup \{C\}) - RSS_T(\mathcal{X}')$ is

$$\sum_{t \in C} \left[(\max(\hat{T}'_t, \lambda_0 + \lambda_C) - T_t)^2 - (\hat{T}'_t - T_t)^2 \right].$$

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

$$\begin{aligned} & |\max(\hat{T}'_t, \lambda_0 + \lambda_C) - T_t| < |\hat{T}'_t - T_t| \\ \Leftrightarrow \quad & \hat{T}'_t < \lambda_0 + \lambda_C < 2T_t - \hat{T}'_t. \end{aligned}$$

The smaller $\hat{T}'_t$ is, the weaker the condition and the greater the number of negative terms. Moreover, a minimal $\hat{T}'_t$ minimizes the negative term, $(\lambda_0 + \lambda_C - T_t)^2 - (\hat{T}'_t - T_t)^2$. Indeed, a consequence of the inequality above is $\hat{T}'_t < T_t$. Algorithms 1 and 3 never remove patterns that were previously added to the model. That is why $\mathcal{X}'$ is necessarily a superset of the model at the current iteration and, given (1), $\forall t \in C$, $\hat{T}'_t \geqslant \hat{T}_t$, where $\hat{T}_t$ is the membership degree that the current model predicts for the n-tuple t. A lower bound of $RSS_T(\mathcal{X}' \cup \{C\}) - RSS_T(\mathcal{X}')$ at the current iteration therefore is

$$\sum_{t \in C \text{ s.t. } \hat{T}_t < \lambda_0 + \lambda_C < 2T_t - \hat{T}_t} \left[(\lambda_0 + \lambda_C - T_t)^2 - (\hat{T}_t - T_t)^2 \right].$$

That lower bound is tight. It is reached when future iterations add to the model patterns at least as dense as C that altogether include every n-tuple $t \in C$ such that $\lambda_C > 2T_t - \hat{T}_t$ and no pattern that modifies the prediction $\hat{T}_t$ of the membership degree of any $t \in C$ such that $\hat{T}_t < \lambda_0 + \lambda_C < 2T_t - \hat{T}_t$.

The patterns in $\mathcal{C}$ are stored in a self-balancing binary search tree that is ordered by increasing lower bound. The initial lower bounds—$\sum_{t \in C \text{ s.t. } 0 < \lambda_C < 2(T_t - \lambda_0)} \left[(\lambda_0 + \lambda_C - T_t)^2 - (\lambda_0 - T_t)^2 \right]$, for all $C \in \mathcal{C}$ —are computed before the first iteration of Algorithm 1 or 3. Algorithm 4 (below) computes $argmin_{C \in \mathcal{C}} RSS_T(\mathcal{X} \cup \{C\})$. Every iteration starts by taking the first pattern C out of $\mathcal{C}$ (line 3). Lines 4–9 compute $RSS_T(\mathcal{X} \cup \{C\}) - RSS_T(\mathcal{X})$ and the lower bound associated with C, given the current model $\mathcal{X}$. Iterations stop (line 2) when $\mathcal{C}$ is empty or, more commonly, when its first pattern is associated with a lower bound that is larger than the smallest difference of RSS_T found so far (lines 11–12). Lines 13–14 reinsert in $\mathcal{C}$ the patterns that were taken out of it and whose lower bounds were updated, except the returned pattern.

Algorithm 4: $\arg\min_{C \in \mathscr{C}} RSS_T(\mathscr{X} \cup \{C\})$

Input: tensor T, tensor $\widehat{T}$ predicted by $\mathscr{X}$, set $\mathscr{C}$ of patterns ordered by increasing lower bound

Output: $\arg\min_{C \in \mathscr{C}} RSS_T(\mathscr{X} \cup \{C\})$

```
1  (min, 𝒰) ← (+∞, ∅)
2  while 𝒞 ≠ ∅ ∧ first lower bound in 𝒞 < min do
3      (lb, C) ← pop 𝒞's first entry
4      (lb, sum) ← (0, 0)
5      forall t ∈ C s.t. T̂_t < λ_0 + λ_C do
6          term ← (λ_0 + λ_C − T_t)^2 − (T̂_t − T_t)^2
7          if term < 0 then
8              lb ← lb + term
9          sum ← sum + term
10     𝒰 ← 𝒰 ∪ {(lb, C)}
11     if sum < min then
12         (min, argmin) ← (sum, C)
13 forall (lb, C) ∈ 𝒰 s.t. C ≠ argmin do
14     insert (lb, C) in 𝒞 respecting the order
15 return argmin
```