ATLANTIS PRESS

Atlantis Studies in Uncertainty Modelling, volume 1

11th Conference of the European Society for Fuzzy Logic and Technology (EUSFLAT 2019)

Bigfoot Climbing the Hill with ILP to Grow Patterns in Fuzzy Tensors

Lucas Maciel and Jônatas Alves and Vinícius Fernandes dos Santos and Loïc Cerf
Computer Science Department, Exact Science Institute, Federal University of Minas Gerais
Avenida Antônio Carlos, 6627, Pampulha, Belo Horizonte, Minas Gerais, Brazil, CEP: 31270-901
lucasmaciel@dcc.ufmg.br, jonatas@dcc.ufmg.br, viniciussantos@dcc.ufmg.br, lcerf@dcc.ufmg.br

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, the most informative patterns according to that model, with high areas times squared densities, are grown by hill-climbing from fragments of them, that a complete algorithm provides. At every iteration, an optimization problem (or its linear relaxation) is solved thanks to *integer linear programming* (or greedily). A *forward selection* then chooses among the discovered patterns a non-redundant subset that fits, but does not overfit, the tensor. Experiments show the proposal discovers high-quality patterns and outperforms state-of-the-art approaches when applied to 0/1 tensors, a special case.

Keywords: Disjunctive box cluster model, Fuzzy tensor, Hill-climbing, Integer Linear Programming, Forward selection.

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 *fuzzy* matrix our analyst may end up with.

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, introduced in 2011 for 0/1 tensors [9], 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 average value 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 present 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 unless the related cell belongs to some pattern. If so, the model predicts that the value

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

	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

Table 2: Fuzzy matrix predicted by the disjunctive box cluster model composed two 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).

is the density of the densest pattern containing the cell. Every pattern density is indeed seen as the *truth value* of the predicate "the regions (the rows of the pattern) like the chocolates (its columns)" and, the disjunctive box cluster model being disjunctive, the fuzzy logic OR operator (i. e., the max operator) is used.

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, the pattern that, alone, best summarizes the tensor is the one with the greatest area (the number of cells it contains) times density squared [9], hence a 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 complete algorithm that provides many small high-density patterns. They are here called *fragments* because the second step consists in growing these fragments, 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, 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, a stepwise regression technique greedily 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 CP decomposition;
- The first pattern-based method to decompose *fuzzy* tensors is presented;
- It discovers high-quality patterns in fuzzy tensors and outperforms state-of-the-art algorithms when applied to 0/1 tensors, a special case;
- Relying on ILP and stepwise-regression, which are not commonly used in the pattern-mining community, the proposal could inspire future research.

2 Basic Definitions

Given $n \in \mathbb{N}$ *dimensions* (i. e., n finite sets, assumed disjoint w.l.o.g.) $D_1, \ldots, D_n$, a *fuzzy tensor* T maps any n-tuple $t \in \prod_{i=1}^{n} D_i$ ($\prod$ 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 $\forall i \in \{1, \ldots, n\}$, $\exists X_i \subseteq D_i \mid X = \prod_{i=1}^{n} X_i$. 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 called a *slice* of S.

3 Related Work

3.1 Complete Algorithms

Given a 0/1 tensor, Data-Peeler [2] list its all-ones sub-tensors that are *closed*, i. e., any strict super-pattern includes an n-tuple with a null membership degree. `multidupehack` [3] generalizes Data-Peeler to fuzzy tensors: n upper-bounds, $\epsilon_1, \ldots, \epsilon_n$, 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, `multidupehack` outputs $X = \prod_{i=1}^{n} X_i$ if $\forall i \in \{1, \ldots, n\}$, $\forall e \in X_i$, $\sum_{t \in X \text{ s.t. } t_i = e} (1 - T_t) \leq \epsilon_i$. It 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". DCE [6] is the only other complete algorithm to mine patterns in fuzzy tensors. Cerf and Meira [3] show that DCE's definition catches patterns that are not sub-patterns of any pattern planted in a synthetic dataset, even if there is little noise affecting that dataset. 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, 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 [11] mines patterns in 3-way tensors and merges them pairwise 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 [4] 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 $\emptyset$ and that to the parent pattern in the dendrogram formalizes the trade-off.

Several algorithms (e. g., [8, 5, 10]) approximately factorize n-way 0/1 tensors. 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 $A^1 \in \{0,1\}^{|D_1| \times r}$, ..., $A^n \in \{0,1\}^{|D_n| \times r}$ 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. `BCP_ALS` [8] heuristically seeks a good CP decomposition of a 0/1 tensor by Alternating Least Squares. DBTF [10] distributes that method on the Spark framework. WALK'N'MERGE [5] 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 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. A fuzzy tensor must be turned 0/1 before applying any algorithm cited in this paragraph.

Non-negative tensor decomposition techniques directly apply but 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 [7] is closest to this work: it substitutes the traditional addition for the max operator and can minimize the Frobenius norm.

4 Disjunctive Box Cluster Model

Mirkin and Kramarenko [9] map pattern mining in 0/1 tensors to a regression problem: 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 $\max_{X \in \emptyset} \lambda_X = 0$, that T_t is

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

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

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

Encoding $\mathcal{X}$ in the Boolean factors A^1, ..., A^n of a rank-$|\mathcal{X}|$ Boolean CP decomposition (see Section 3.2), $RSS_T(\mathcal{X}) = \|T - \lambda_0 \mathbb{1} - \max_{k=1}^{|\mathcal{X}|} \lambda_k A^1_{:k} \otimes \cdots \otimes A^n_{:k}\|^2$. 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 [9] repeatedly searches for one single pattern X that locally minimizes $X \mapsto RSS_T(\{X\})$. 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 final pattern is added to $\mathcal{X}$, the model.

5 `Bigfoot`

Like *TriclusterBox* [9], the algorithm proposed in the present article uses hill-climbing to repeatedly search for one pattern that locally minimizes the residual sum of squares (2) of Model (1), i. e., that locally maximizes $g : X \mapsto |X|\lambda_X^2$ with $\lambda_X = \frac{\sum_{t \in X}(T_t - \lambda_0)}{|X|}$. However, the *locality* is differently defined. *TriclusterBox* considers two patterns $X = \prod_{i=1}^{n} X_i$ and $Y = \prod_{i=1}^{n} Y_i$ neighbors 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, X and Y are neighbors if and only if they include the initial pattern and $|\cup_{i=1}^{n} X_i| - |\cup_{i=1}^{n} Y_i| = \pm 1$. Unless X is the initial pattern, X therefore has more neighbors, the pattern space is more thoroughly explored and a higher local maximum of g is reached.

`multidupehack` [3], presented in Section 3.1, can be used to compute from a fuzzy tensor T a set $\mathcal{F}$ of initial patterns to grow. They are called *fragments* from now on. T and $\mathcal{F}$ are the input of Algorithm 1, named `Bigfoot`[1]. It outputs a set $\mathcal{C}$ of patterns that are can-

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

didate explanatory variables for the disjunctive box cluster model. Line 4 selects the fragment F that best complements the previously discovered patterns, i. e., that minimizes $G \mapsto RSS_T(\mathcal{C} \cup \{G\})$, and that is not included in any of them (line 18). Computing F is not as expensive as it looks. At a given iteration of Algorithm 1, $RSS_T(\mathcal{C})$ is a constant. As a consequence, $F = \arg\min_{G \in \mathcal{F}}(RSS_T(\mathcal{C} \cup \{G\}) - RSS_T(\mathcal{C}))$. 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 $RSS_T(\mathcal{C} \cup \{G\}) - RSS_T(\mathcal{C})$ 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 $O(\sum_{G \in \mathcal{F}} |G|)$.

Algorithm 1: `Bigfoot`

Input: fuzzy tensor T, fragment set $\mathcal{F}$
Output: patterns in T that locally maximize g

```
1  C ← ∅
2  F_∪ ← ∪_{F∈F} F
3  while F ≠ ∅ do
4      F ← arg min_{G∈F} RSS_T(C ∪ {G})
5      X^max ← F
6      repeat
7          optimal ← true
8          (σ_1, ..., σ_n) ← (|X_1^max|, ..., |X_n^max|)
9          for i = 1 → n do
10             σ_i ← σ_i + 1
11             X ← f(T, F_∪, F, g(X^max), σ_1, ..., σ_n)
12             if g(X) > g(X^max) then
13                 X^max ← X
14                 optimal ← false
15             σ_i ← σ_i − 1
16     until optimal
17     C ← C ∪ {X^max}
18     F ← {F ∈ F | F ⊈ X^max}
19 return C
```

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 number of elements $(\sigma_1, \ldots, \sigma_n) \in \mathbb{N}^n$ 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 $X^{\max}$ evaluates g to a value exceeding $g(X^{\max})$), $X^{\max}$'s neighbor maximizing g became the best pattern so far (line 13) and the optimization goes on.

Algorithm 1's efficiency mainly depends on the time it takes to find the pattern $X = \prod_{i=1}^{n} X_i$, among the super-patterns of F with $|X_1| = \sigma_1, \ldots, |X_n| = \sigma_n$, that maximizes g. The function f, called at line 11, computes X by Integer Linear Programming (ILP). The problem is expressed as the maximization of

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

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

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

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

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

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

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., $X \supseteq F$. Constraint (6) forces the returned pattern $X = \prod_{i=1}^{n} X_i$ to have $\forall i \in \{1, \ldots, n\}, |X_i| = \sigma_i$. Given those constant cardinalities, $|X|$ is constant too. It is $\prod_{i=1}^{n} \sigma_i$. That is why maximizing $g(X) = |X| \lambda_X^2 = \frac{(\sum_{t \in X}(T_t - \lambda_0))^2}{|X|}$ 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 found 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: 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, Constraint (4)

equates to $\forall t \in \mathcal{F}_\cup$, $w_t = 1 \Leftrightarrow \forall i \in \{1, \ldots, n\}$, $x_{t_i} = 1$ and a valuation of the x_e variables implies a unique valuation of the (far more numerous) w_t variables.

Finally, the following constraint does not alter Algorithm 1's output but is added for efficiency reasons:

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

$\prod_{i=1}^{n} \sigma_i$ 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.

6 `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 its linear relaxation, 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) forces $0 \leq \sum_{i=1}^{n} x_{t_i} - n w_t$. 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

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

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

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

The innermost sums are constants. They only need to be computed once, before running Algorithm 1. A greedy algorithm solves, in an exact way, Problem (10) under the (remaining) constraints (5), (6) and (8). Constraint (5) forces $x_e = 1$ for all elements e involved in the fragment F. Additional x_e variables are set to 1 to satisfy Constraint (6): those with e in the proper dimension and the greatest $\sum_{t \in \mathcal{F}_\cup \text{ s.t. } t_i = e} (T_t - \lambda_0)$.

`Bigfoot-LR` is the name given to Algorithm 1 when f solves, in $O(\sum_{i=1}^{n} \sigma_i)$ time, Problem (10). However, only growing the fragments with n-tuples in the densest slices of $\mathcal{F}_\cup$ is naive. Section 8 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 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.

7 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. Stepwise regression selects a statistically more relevant subset of those explanatory variables, i. e., a simpler model whose residual sum of squares is not significantly higher. The search of a trade-off between the goodness of fit and the simplicity (how many patterns) is formalized as the minimization of a measure, e. g., the *Akaike information criterion* (AIC) [1]. 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 an 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 an 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).

Algorithm 2 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: it takes $O(\sum_{C \in \mathcal{C}} |C|)$-time and favors the selection of a pattern with little intersection with the previously selected patterns because the predicted membership degrees for the n-tuples in this 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 2: `forward-select`

Input: fuzzy tensor T, candidate pattern set $\mathcal{C}$
Output: $\mathcal{X} \subseteq \mathcal{C}$ fitting T through Model (1)

$\mathcal{X} \leftarrow \emptyset$
while $\mathcal{X} \neq \mathcal{C}$ **do**
  $X \leftarrow \arg\min_{C \in \mathcal{C}} RSS_T(\mathcal{X} \cup \{C\})$
  if $AIC_T(\mathcal{X} \cup \{X\}) > AIC_T(\mathcal{X})$ **then**
    return `forward-select`$(T, \mathcal{X})$
  $\mathcal{X} \leftarrow \mathcal{X} \cup \{X\}$
return $\mathcal{X}$

8 Experimental Validation

GCC 5.4.1 is used to compile implementations in C++ of `multidupehack` [3], `Bigfoot`, `Bigfoot-LR` and *TriclusterBox* [9], all distributed under the terms of the GNU GPLv3. Walk'n'Merge [5] was provided by its authors. DBTF [10] is only distributed in binary form. All experiments are performed on a GNU/Linux system running on top of 2.4 GHz cores, 12 MB of cache and 32 GB of RAM. `Bigfoot` uses IBM ILOG CPLEX Optimizer v12.6.0 [2] to solve the ILP problems. `https://gitlab.com/lucasmaciel82/Bigfoot` hosts the datasets, the source codes and the scripts.

8.1 Real-World Tensor

The 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 13^{th} to April 6^{th}). 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 turns each normalized number of retweets into an influence, in $[0, 1]$. Summing all influences gives 40,167.3, i. e., $\frac{40{,}167.3}{170{,}670 \times 29 \times 12} \approx 0.0006$ times the maximal possible value. Since $\lambda_0 = 0.0006$ 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` [3] provides the initial fragments. Three minimal size constraints are enforced (see Section 3.1): at least eight users, two teams and four weeks. With the upper-bounds $\epsilon_{\text{team}} = 0.5$, $\epsilon_{\text{team}} = 2$ and $\epsilon_{\text{week}} = 1$ (again, see Section 3.1), `multidupehack` takes 1.1 second to provide 359 fragments. Given their minimal sizes and the upper-bounds, any slice of any fragment has, at least, a 0.875 density. 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 2 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 last column of Table 3 gives the densities of the patterns. The patterns are large and dense.

$\lvert X_{\text{user}}\rvert$	$\lvert X_{\text{weeks}}\rvert$	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

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

The first four patterns do not intersect, a property favored by the forward selection, as explained in the last paragraph of Section 7. 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 identifiable users involved in a given pattern are either journalists, who are indeed influential, or supporters of one of the teams in the pattern. Within 2.7 seconds, `Bigfoot-LR` grows the same 359 fragments into 46 slightly larger fragments.

TriclusterBox [9], Walk'n'Merge [5] and DBTF [10] 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 4). 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.

[2] `https://www.ibm.com/analytics/cplex-optimizer/`

8.2 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.

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 fixing the number of incorrect observations to 0 tune the level of noise applied by *inverse transform sampling*. More correct observations lead to membership degrees that are closer to the values 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

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

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 results report as well the number of patterns that are discovered, $|\mathcal{X}|$.

Figure 1 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` [3] provides the fragments, with at least three elements of every dimension and $\epsilon_1 = \epsilon_2 = \epsilon_3 = 2.7$. In this way, the minimal possible density for any slice of a fragment is $\frac{3^2 - 2.7}{3^2} = 0.7$. Forward selection (see Section 7) 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. 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. It recovers all four planted patterns, except in the noisiest setting (1 correct observation), where the quality of the discovered patterns still exceeds 0.9. `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. `Bigfoot-LR`+`Bigfoot` is faster than `Bigfoot`, except when it grows patterns in the noisiest setting. The end of Section 6 explains why that can happen.

`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 7, always improves its output. *TriclusterBox*'s results in Figure 2 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. Here, that means any slice of any fragment contains at most two 3-tuples with null membership degrees. Figure 2 shows that `Bigfoot` does not always reach the qualities reported in Figure 1: 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.

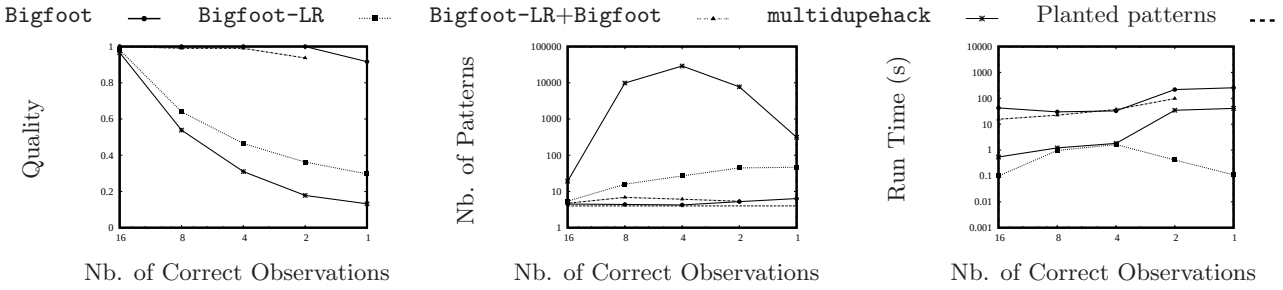

Figure 1: Qualities, numbers of patterns and run times of `multidupehack`, `Bigfoot`, `Bigfoot-LR` and `Bigfoot-LR+Bigfoot` mining 3-way fuzzy tensors (the level of noise increases from left to right)

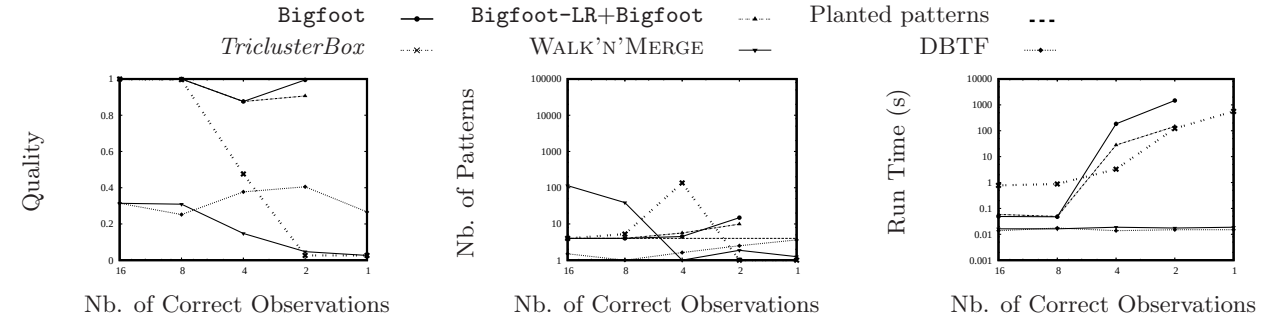

Figure 2: Qualities, numbers of patterns and run times of the competing methods mining noisy 3-way 0/1 tensors

9 Conclusion

This article has presented the first solution to a generalization of Boolean CP tensor factorization: the decomposition of *fuzzy* tensors into patterns weighted by their densities. The proposal combines various techniques, such as hill-climbing, integer linear programming and stepwise regression, to grow pattern fragments and select a small number of grown patterns. The method successfully recovers patterns in noisy tensors. It even outperforms state-of-the-art approaches when the tensor is 0/1, a special case.

Acknowledgement

The work has been partially funded by the FAPEMIG under Grant № APQ-04224-16.

References

[1] H. Akaike, A new look at the statistical model identification, Transactions on Automatic Control 19 (6) (1974) 716–723.

[2] L. Cerf, J. Besson, C. Robardet, J.-F. Boulicaut, Closed patterns meet n-ary relations, ACM Transactions on Knowledge Discovery from Data 3 (1) (2009) 1–36.

[3] L. Cerf, W. Meira Jr., Complete discovery of high-quality patterns in large numerical tensors, in: Proc. IEEE ICDE'14, 2014, pp. 448–459.

[4] L. Cerf, P.-N. Mougel, J.-F. Boulicaut, Agglomerating local patterns hierarchically with ALPHA, in: Proc. ACM CIKM'09, 2009, pp. 1753–1756.

[5] D. Erdős, P. Miettinen, Walk'n'Merge: A scalable algorithm for boolean tensor factorization, in: Proc. IEEE ICDM'13, 2013, pp. 1037–1042.

[6] E. Georgii, K. Tsuda, B. Schölkopf, Multi-way set enumeration in weight tensors, Machine Learning 82 (2) (2011) 123–155.

[7] S. Karaev, P. Miettinen, Cancer: Another algorithm for subtropical matrix factorization, in: Proc. ECML PKDD'16, 2016, pp. 576–592.

[8] P. Miettinen, Boolean tensor factorizations, in: Proc. IEEE ICDM'11, 2011, pp. 447–456.

[9] B. G. Mirkin, A. V. Kramarenko, Approximate bicluster and tricluster boxes in the analysis of binary data, in: Proc. RSFDGrC'11, 2011, pp. 248–256.

[10] N. Park, S. Oh, U. Kang, Fast and scalable distributed boolean tensor factorization, in: Proc. IEEE ICDE'17, 2017, pp. 1071–1082.

[11] L. Zhao, M. J. Zaki, triCluster: An effective algorithm for mining coherent clusters in 3D microarray data, in: Proc. ACM SIGMOD'05, 2005, pp. 694–705.