Complete Discovery of High-Quality Patterns in Large Numerical Tensors

Loïc Cerf [1], Wagner Meira Jr. [2]

Department of Computer Science, Universidade Federal de Minas Gerais
Belo Horizonte, Brazil
[1] lcerf@dcc.ufmg.br
[2] meira@dcc.ufmg.br

***Abstract*—Many datasets are numerical tensors, i. e., associate n-tuples with numerical values. Until recently, the discovery of relevant local patterns in such numerical and multidimensional data has received little attention despite the broad applicative perspectives offered by this general framework. Even in the simpler 2-dimensional case, almost every proposal so far is either incomplete (i. e., it does not list *every* pattern) or relies on binning and mines Boolean tensors. In both cases, some information is lost during the process. In uncertain tensors, n-tuples satisfy the studied predicate to a certain extent and no information is lost w.r.t. the original data. Given an uncertain tensor, the closed patterns are its maximal "sub-tensors" covering n-tuples that "mostly" satisfy the predicate. Defining "mostly" is the key problem: the patterns should be both relevant given the data and efficiently extractable. The proposed complete extractor reuses the enumeration principles of the state-of-the-art miner for closed n-sets but incrementally enforces the newly designed definition. In this way, the proposed algorithm runs orders of magnitude faster than its only competitor and large datasets are tractable. The experimental section reports the discovery of dynamic patterns of influence in Twitter as well as usage patterns in a transportation network. Additional experiments on synthetic data quantitatively assess the quality of the chosen definition for the patterns.**

I. Introduction

The patterns considered in this paper are numerical sub-tensors that satisfy relevance constraints such as minimal sizes and a tolerance to noise in the input data. An application scenario (described in details in Section V-B1), which benefits from such patterns, is that of analyzing Twitter messages about the Brazilian soccer championship. The goal is to discover sets of users who were, during some weeks, influential when they were writing about some specific teams. How many times a user is retweeted (i. e., other users "repeat" her messages) is known to be a good measure of her influence [1]. Considering these numbers for every user, week and commented team, the analyst faces a 3-dimensional numerical tensor. Figure 1 depicts a sample of it. Each of its values represents, for a given week, the total number of retweets a user received when writing about a team. Let us assume that the analyst wants to list *every* large sub-tensor (that is, an incomplete or approximate solution is not acceptable) that associates a subset of users with a subset of teams and a subset of weeks, and that covers high quantities of retweets. For instance, in Figure 1, she wants to discover that the users `globoesportecom` and `milton_neves` (and *not* `oclebermachado`) were influential during the weeks `47` and `48` when they were writing about `Fluminense`, `Santos` and `São Paulo` (and *not* `Ceará`). Indeed, most of the values in this sub-tensor are higher than 20, a threshold the analyst considers appropriate to separate between "being influential" and "not being influential".

By using a *crisp* threshold of 20 retweets, the analyst would not discover the pattern mentioned above because, during the week `48`, `milton_neves`' messages about `Santos` were retweeted only 19 times. Accepting upper-bounded quantities of values below 20 may be a solution. However, the patterns should obviously be more tolerant with values just below the threshold than with values clearly below it. This is about using a *fuzzy* threshold. In other terms, the predicate "being influential", taking a number of retweets as argument, returns a value *between* 0 and 1 (0 meaning "not influential" and 1 "influential") and not 0 *or* 1. Figure 2 depicts a classical example of such a function. Because it is bijective, no information is lost w.r.t. the input numerical data. Choosing and evaluating the fuzzy predicate leads to a so-called uncertain tensor and the actual starting point of the pattern mining problem.

The mere definition of the patterns of interest is the first challenge this paper tackles. A pattern must tolerate, to some extent, the inclusion of values that do not fully satisfy the fuzzy predicate, i. e., that do not evaluate it to 1. This is about noise tolerance. The question is: how to define it? This paper argues, both theoretically and empirically, in favor of a *per-element* and *absolute* tolerance to noise. This means a sub-tensor is a pattern to return only if *each* of its hyperplanes (related to single elements of the dimensions) tolerate an upper-bounded *total* quantity of noise (the noise affecting the values is summed and not averaged). This contrasts with the *per-pattern* and *relative* tolerance to noise that is matched by the DCE algorithm [2], [3], which is, to the best of our knowledge, the only previous work that tackled the complete extraction of "itemset-like" patterns in numerical tensors. Comparing the two approaches, this paper shows and explains why 1) a *per-element* tolerance to noise entails the discovery of patterns of higher quality and 2) the *absolute* aspect allows an enumeration of the patterns that is orders of magnitude faster than DCE's. Besides, and those are other contributions of this paper, our choice of a definition makes it possible to losslessly condense the collection of patterns into the most informative ones and to expressively constrain these patterns

	Ceará	**Fluminense**	**Santos**	**São Paulo**	Ceará	**Fluminense**	**Santos**	**São Paulo**
globoesportecom	0	**109**	**23**	**146**	0	**77**	**57**	**47**
oclebermachado	0	104	0	2	0	381	0	0
milton_neves	0	**195**	**68**	**21**	0	**143**	**19**	**22**
	week 47				**week 48**			

Fig. 1. Part of the 3-dimensional tensor giving, week after week, the total number of retweets users receive when writing about the different Brazilian soccer teams. A relevant pattern covers the values in bold.

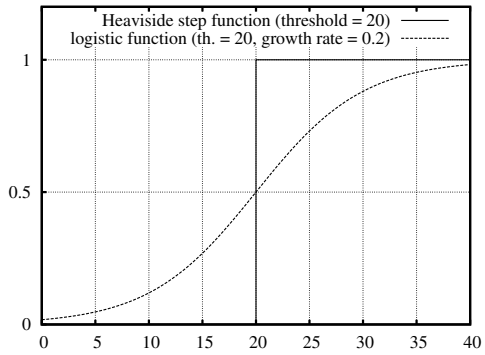

Fig. 2. To be turned Boolean, a numerical measure goes through a Heaviside step function. Being able to mine uncertain data allows to replace this "all or nothing" function by, typically, a logistic one, which is bijective hence no loss of information.

so that they are both more relevant and extracted faster.

After presenting the related work in Section II, Section III gives and theoretically argues for our definition of a pattern in a numerical tensor. Section IV deals with their efficient and possibly constrained extraction. The actual algorithm is named `multidupehack`[1] and experiments in Section V empirically confirm that it outperforms DCE in terms of quality and running times. Section VI sums up the advancement in our understanding of how noise should be tolerated and emphasizes the broad applicability of `multidupehack`. Finally, Section VII briefly concludes.

II. Related Work

The work on the complete though efficient extraction of *closed* itemsets in Boolean matrices started in the late 90s with the publications of [4] and [5]. Since then, several generalizations toward noise tolerance (also known as "fault-tolerance") have been proposed. The complete approaches usually still consider Boolean data (see [6] for a survey), i. e., the defined patterns tolerate upper-bounded quantities of '0' values. To the best of our knowledge, FTFIM [7] is the most efficient proposal so far. The generalization of closed itemset mining toward Boolean tensors is more recent. CubeMiner [8] and Trias [9] are restricted to 3-dimensional Boolean tensors. In contrast, Data-Peeler [10] can mine Boolean tensors of any dimensionality. Despite its greater applicability, Data-Peeler runs orders of magnitude faster than both CubeMiner and Trias on 3-dimensional tensors. Fenster [11] generalizes Data-Peeler. It still only processes Boolean tensors but tolerates upper-bounded quantities of '0' values per hyperplane of the closed patterns. CGQBminer [12] completely lists noise tolerant closed patterns in 3-dimensional Boolean tensors. These patterns were baptized *maximal cross-graph quasi-bicliques* by analogy with *maximal cross-graph quasi-cliques* [13], [14]. Indeed, the underlying noise tolerance is similar: in *every* (bipartite) graph involved in the pattern, *every* vertex of a quasi-(bi)clique is mostly connected with the others vertices. The dimension gathering graph labels plays a particular role in this definition. Furthermore, the constraints it imposes on the patterns are quite numerous (one per couple (vertex,graph) in the pattern). CACTUS [15] and Clicks [16] list the so-called *maximal subspace clusters* in n-dimensional Boolean tensors. In such patterns, which do not necessarily involve all n dimensions, the noise is tolerated in an even more constrained way: every pair of elements must be *strongly connected*.

The approaches referenced above only process Boolean data. triCluster [17] mines 3-dimensional numerical tensors that do not translate to fuzzy data, i. e., to the weighted satisfaction of a predicate. Indeed, the proposed type of pattern, the *maximal coherent clusters*, captures aspects such as similarity, shifting or scaling of the covered values. [18] additionally allows to simultaneously specify a shifting and a scaling coherence. Starting with [19], some works use a probabilistic framework to exactly compute the expected frequencies of the itemsets in uncertain matrices. These approaches suffer from significant scalability issues and it was recently shown that those frequencies are very accurately approximated and much more efficiently computed by sampling [20] or with a weak version of the central limit theorem [21]. In all cases, the discovered patterns are 1-dimensional itemsets associated with frequency measures. DCE [3], [2] is, to the best of our knowledge, the only proposal so far that completely extracts patterns in uncertain tensors. Because its objectives are similar to ours, DCE is taken as a reference all along this paper.

III. Definition

All along this section, $\times$ denotes the Cartesian product and $\prod$ is used for the Cartesian product of more than two sets. Given the $n \in \mathbb{N}$ dimensions (i. e., n finite sets) $(D_i)_{i=1..n}$ of the numerical dataset, the actual pattern mining problem is defined on the *uncertain tensor* $T \in [0,1]^{\prod_{i=1}^{n} D_i}$ resulting from the transformation of every numerical value into a degree of satisfaction of the considered fuzzy predicate. A pattern in this tensor associates subsets of each of the n dimensions, i. e., is in $\prod_{i=1}^{n} 2^{D_i}$. The pattern $(X_i)_{i=1..n} \in \prod_{i=1}^{n} 2^{D_i}$ is said to

[1] `multidupehack` is not a dupe hack. It stands for MULTI-Dimensional Uncertain Pattern Extractions Having A Closedness Knowledge.

	a	b	c	d
α	0.4	1	1	0
β	1	1	1	0
γ	1	1	1	0

Fig. 3. A 2-dimensional pattern that "globally" tolerates noise.

cover the n-tuple $t \in \prod_{i=1}^{n} D_i$ if $t \in \prod_{i=1}^{n} X_i$.

A. Noise Tolerance

To be relevant, a pattern should mainly cover n-tuples having, in T, values close to 1. Considering, *a posteriori*, that the pattern "holds" in the dataset, the differences to one of the covered values are *noise*, i. e., unwanted alterations of the dataset. The following function takes a "sub-tensor" as an argument and returns the (absolute) quantity of noise it contains:

Definition 1: Given an uncertain tensor $T \in [0,1]^{\prod_{i=1}^{n} D_i}$,

$$q : \prod_{i=1}^{n} 2^{D_i} \to \mathbb{R}_+$$
$$(X_i)_{i=1..n} \mapsto \sum_{t \in \prod_{i=1}^{n} X_i} 1 - T_t$$

, where T_t is the membership degree of the n-tuple t in T.

The absolute/relative dilemma will be discussed later and has no influence on a more fundamental decision to be taken: what subspace(s) of a pattern should be given as an argument of q (or its relative counterpart) for its/their noise to be controlled? Two "extreme" answers come to mind. The first one would simply state that *every* n-tuple covered by a pattern should have a value exceeding a threshold, i. e., q would be given individual n-tuples at input. This actually amounts to turning the tensor T Boolean, i. e., to replacing its values in the $[0,1]$ interval by values in the $\{0,1\}$ set. The introduction of this paper has already mentioned the loss of information related to this choice. The second answer that comes to mind would consider the noise covered by a *whole* pattern, i. e., q would be given a whole pattern at input. Unfortunately, patterns that globally cover an upper-bounded quantity of noise often are irrelevant. Figure 3 depicts the issue. It represents the restriction of a 2-dimensional dataset to the pattern $(\{\alpha, \beta, \gamma\}, \{a, b, c, d\})$. This pattern "globally" tolerates a quantity 3.6 (or 30%) of noise. Nevertheless d should obviously not be part of the pattern.

Liu et al. [22] have thoroughly studied this issue and shown that the noise should be bounded *per-element* in the pattern. Their definition is restricted to Boolean matrices, i. e., specifies maximal quantities of '0' values that are tolerated per row and column in a 2-dimensional pattern. In a tensor, these hyperplanes become "slices". The following function takes as arguments a pattern and an element; it returns the projection of the pattern on the element. If the element is in the pattern, it therefore outputs the "slice" of the pattern related to the element.

Definition 2: Given an uncertain tensor $T \in [0,1]^{\prod_{i=1}^{n} D_i}$,

$$h : \prod_{i=1}^{n} 2^{D_i} \times \cup_{i=1..n} D_i \to \prod_{i=1}^{n} 2^{D_i}$$
$$((X_i)_{i=1..n}, x) \mapsto (X_1, \ldots, \{x\}, \ldots, X_n)$$

In an uncertain tensor, a per-element tolerance to noise is about defining upper-bounded quantities of noise per slice of the pattern. These upper-bounds can be different from a dimension to another so that quantifying the noise in an absolute way does not prevent from discovering patterns with many more elements in a dimension than in another.

Definition 3: Given an uncertain tensor $T \in [0,1]^{\prod_{i=1}^{n} D_i}$ and n thresholds $(\epsilon_i)_{i=1..n} \in \mathbb{R}_+^n$, a pattern $X \in \prod_{i=1}^{n} 2^{D_i}$ is *connected* if and only if $\forall i = 1..n$, $\forall x \in X_i$, $q(h(X, x)) \leq \epsilon_i$.

Thanks to this definition, any element in a connected pattern "really" belongs to the pattern. For instance, the pattern $(\{\alpha, \beta, \gamma\}, \{a, b, c, d\})$ in Figure 3 would not be discovered because d covers a quantity 3 of noise and the analyst would certainly not set such a high tolerance to noise per column (Latin letters) to discover patterns with as few as 3 rows. On the contrary, $(\{\alpha, \beta, \gamma\}, \{a, b, c\})$ is discovered if both ϵ_1 and ϵ_2 exceed 0.6, which is the highest quantity of noise covered by a row (α) and a column (a) in the pattern. Let us mention that DCE mines patterns that "globally" tolerate noise. Since that algorithm relies on the reverse search technique [23], a "per-element" tolerance, like that of Definition 3, cannot be enforced along the enumeration. Admitting the relevance problem illustrated by Figure 3, the authors of DCE have proposed, in [2], a post-processing step selecting the patterns satisfying such a definition.

Two reasons justify the choice of an *absolute* tolerance to noise: a declarative reason and a procedural one. Both are related to the following theorem, known in the literature as a *downward closure*, and stating that the "sub-patterns" of a connected pattern are connected as well.

Theorem 1: If $(X_i)_{i=1..n} \in \prod_{i=1}^{n} 2^{D_i}$ is connected then $\forall (X'_i)_{i=1..n} \in \prod_{i=1}^{n} 2^{D_i}$, $\Big(\forall i = 1..n,\ X'_i \subseteq X_i\Big) \Rightarrow (X'_i)_{i=1..n}$ is connected.

Its proof relies on the fact that, in Definition 3, removing positive terms decreases the sums, which therefore remain below the noise tolerance thresholds. Theorem 1 means that an exhaustive search for connected patterns does not have to traverse the "super-patterns" of the unconnected ones, i. e., an "Apriori trick" [24] can be safely used to perform significantly faster. For instance, in Figure 3, the element α covers a quantity 1.6 of noise in the pattern $(\{\alpha\}, \{a, d\})$. If that exceeds the tolerance chosen by the analyst, the "super-patterns" (i. e., those having α, a, d and at least one other Greek or Latin letter) need not be considered. Indeed, the noise covered by their element α would only increase and remain above the tolerated threshold. With a relative tolerance to noise, like that of DCE, Theorem 1 does not hold anymore. Instead, it can only be proved that, given a (relatively) connected pattern, *there exists* a (relatively) connected "sub-pattern" with one

element less. This existential quantification (instead of the universal one in Theorem 1) prevents the sound use of an "Apriori trick". Instead, DCE relies on the reverse search technique [23] and runs orders of magnitude slower than `multidupehack` (see Section V).

B. Closedness

Complete collections of patterns satisfying Definition 3 usually are very redundant. Indeed, according to Theorem 1, the strict "sub-patterns" of a connected pattern are connected too. Nevertheless, those "sub-patterns" do not bring any added-value to the collection. Indeed, like in Boolean matrices [25], the most informative patterns are the closed ones, i. e., those that cannot be enlarged without becoming unconnected.

Definition 4: A connected pattern $(X_i)_{i=1..n} \in \prod_{i=1}^{n} 2^{D_i}$ is *closed* if and only if $\forall (X'_i)_{i=1..n} \in \prod_{i=1}^{n} 2^{D_i}$,

$$\begin{cases} \forall i = 1..n,\ X_i \subseteq X'_i \\ (X'_i)_{i=1..n} \text{ is connected} \end{cases} \Rightarrow (X'_i)_{i=1..n} = (X_i)_{i=1..n}.$$

According to Theorem 1, the closed patterns are a lossless condensed representation [26] of the connected patterns, i. e., the latter collection can be deduced from the closed patterns only. It contains all their "sub-patterns" and only them. In contrast, a relative tolerance to noise does not make the closed patterns be a lossless condensed representation. The reason for that is, again, the absence of a downward closure in a relative framework. For instance, in Figure 3, the noise per column (Latin letters) in $(\{\alpha, \beta, \gamma\}, \{a, b, c\})$ never exceeds 20%. However the "sub-patterns", that still cover (α, a) but with strictly less rows (Greek letters), all exceed this threshold. Because DCE tolerates noise in a relative way, it cannot force the extracted patterns to be closed: the information carried by all connected patterns would be partly lost. This explains why DCE lists larger collections of patterns than those discovered with `multidupehack` (see Section V-A2), i. e., the closed patterns as defined by Definitions 3 and 4.

To enforce the closedness, `multidupehack` actually considers smaller and smaller patterns. In this way, when a pattern is proved unclosed (i. e., can be enlarged by an element absent from it), `multidupehack` does not have to traverse its "sub-patterns", which can be enlarged by the same element. This is, again, about using an "Apriori trick" for faster extractions. However, it only is effective if the closedness definition is extended to *any* pattern (Definition 4 only applies to *connected* patterns). That is why the following definition is used as a replacement for Definition 4.

Definition 5: Given an uncertain tensor $T \in [0,1]^{\prod_{i=1}^{n} D_i}$ and n thresholds $(\epsilon_i)_{i=1..n} \in \mathbb{R}_+^n$, a pattern $(X_i)_{i=1..n} \in \prod_{i=1}^{n} 2^{D_i}$ is *closed* if and only if $\forall i = 1..n$, $\forall s \in D_i \setminus X_i$,

$$\begin{cases} (1)\ q(h(X, s)) > \epsilon_i \\ \text{or} \\ (2)\ \exists x \in \cup_{j \neq i} X_j \mid \\ q(h((X_1, \ldots, X_{i-1}, X_i \cup \{s\}, X_{i+1}, \ldots, X_n), x)) > \epsilon_j \end{cases}.$$

This definition is equivalent to Definition 4 when the considered pattern is connected. Since Definition 5 only deals with enlargements by *one* element, proving this equivalence relies on Theorem 1. It is interesting to understand, in Definition 5, the two verifications that must be done when attempting the enlargement of a pattern with an element s absent from it. For this pattern to be closed, (1) its projection on s must contain too much noise or (2) s must introduce noise on an orthogonal element (i. e., an element in another dimension) of the pattern and make it exceed the tolerated threshold. For instance, with $\epsilon_1 = \epsilon_2 = 0.75$, the pattern $(\{\alpha, \beta, \gamma\}, \{a, b, c\})$ in Figure 3 is not extensible with d either because (1) d covers too much noise, 3, on the rows α, β and γ or because (2) α covers too much noise, 1.6, on the columns a, b, c and d.

IV. Complete Extraction

A. Data-Peeler's Enumeration

The closed patterns in uncertain tensors are *syntactically* identical to those in Boolean tensors: they are in $\prod_{i=1}^{n} 2^{D_i}$. Leaving aside the small but important differences described in Sections IV-D and IV-E, `multidupehack` traverses the pattern search space in the same efficient way as Data-Peeler, the state-of-the-art closed pattern extractor in Boolean tensors. That is why Algorithm 1[2] is similar to the one that describes Data-Peeler's enumeration in [10]. Due to lack of space, the underlying principles cannot be thoroughly described here. Let us just stress that the search space is recursively explored depth-first and that, at every iteration, U lower-bounds it (i. e., every element in U is in the patterns that will be recursively considered) and is used to enforce the connection (see Definition 3), whereas $U \cup V$ upper-bounds it (i. e., every element in the recursively considered patterns is in $U \cup V$) and is used to enforce the closedness (see Definition 5). The third set, S, gathers elements that were previously enumerated but refused (second recursive call). They happen to be the only ones that may enlarge a recursively discovered pattern, i. e., make it unclosed. Given these scarce explanations, it should be clear that `multidupehack` is initially called with $(U_0, V_0, S_0) = (\emptyset, \cup_{i=1}^{n} D_i, \emptyset)$.

B. Incremental Verification of the Connection and the Closedness

`multidupehack` shares the enumeration principles of Data-Peeler but is not a trivial extension of it. The connection and closedness tests require much more work. A naive verification of Definition 3 (resp. Definition 5) would access all fuzzy values in T that are associated with the n-tuples covered by the tested pattern (resp. the pattern augmented with the elements in S, which may prevent its closedness). That would lead to disastrous running times. To remain tractable on large tensors, `multidupehack` relies on the following observation of Algorithm 1: between two recursive calls, the search-space bounds, U and $U \cup V$, do not change much. U only grows by one element after the first recursive call and is left unchanged after the second one. $U \cup V$ loses one element after the second recursive call; potentially more after the first one. To avoid

[2]To simplify the notations, in Algorithm 1 and all along Section IV, a pattern, $(X_i)_{i=1..n}$, and its set of elements, $\cup_{i=1}^{n} X_i$, are mixed up.

Algorithm 1 `multidupehack`.

Data: U, V, S
Output: Every closed pattern containing every element in U, possibly some elements in V, and satisfying a piecewise (anti)-monotone constraint $\mathcal{C}$ (see Section IV-F)
if $\mathcal{C}$ may be satisfied by a pattern descending from this node $\wedge U \cup V$ is closed **then**
  if $V = (\emptyset, \dots, \emptyset)$ **then**
    $\mathbf{output}(U)$
  else
    Choose $e \in V$
    `multidupehack`$(U \cup \{e\},$
    $\{v \in V \setminus \{e\} \mid U \cup \{e\} \cup \{v\}$ is connected$\},$
    $\{s \in S \mid U \cup \{e\} \cup \{s\}$ is connected$\})$
    `multidupehack`$(U, V \setminus \{e\}, S \cup \{e\})$
  end if
end if

repetitive scans of the same parts of T, `multidupehack` incrementally updates "noise counters" in the subspaces of T that are relevant to the verifications of Definitions 3 and 5 on the recursively considered patterns. In this way, at every recursive call, `multidupehack` only needs to access the parts of T related to the symmetric differences of U (resp. $U \cup V$) before and after the call. Writing the definitions of the connection (Definition 3) and the closedness (Definition 5) in the contexts in which they are verified in Algorithm 1 gives the following relevant counts of noise:

- To check the closedness of $U \cup V$:
  - $\forall s \in S$, $q(h(U \cup V, s))$;
  - $\forall s \in S$, $\forall u \in U$, $q(h(U \cup V \cup \{s\}, u))$.
- Given $e \in V$ and $U_{\text{first}} = U \cup \{e\}$ (the elements that are present in every pattern recursively considered after the first call of `multidupehack` in Algorithm 1), to compute $\{v \in V \setminus \{e\} \mid U_{\text{first}} \cup \{v\}$ is connected$\}$:
  - $\forall v \in V, q(h(U_{\text{first}}, v))$;
  - $\forall v \in V, \forall u \in U_{\text{first}}, q(h(U_{\text{first}} \cup \{v\}, u))$.
- Given $e \in V$ and $U_{\text{first}} = U \cup \{e\}$, to compute $\{s \in S \mid U_{\text{first}} \cup \{s\}$ is connected$\}$:
  - $\forall s \in S, q(h(U_{\text{first}}, s))$;
  - $\forall s \in S, \forall u \in U_{\text{first}}, q(h(U_{\text{first}} \cup \{s\}, u))$.

By factorizing the last two points, four families of counters are useful:

- $\forall f \in S, q(h(U \cup V, f))$;
- $\forall f \in S, \forall u \in U, q(h(U \cup V \cup \{f\}, u))$;
- $\forall f \in V \cup S, q(h(U_{\text{first}}, f))$;
- $\forall f \in V \cup S, \forall u \in U_{\text{first}}, q(h(U_{\text{first}} \cup \{f\}, u))$.

During the recursion, any element in V may be moved to U or S. By keeping updated every type of counters ($q(h(U \cup V, f))$, $q(h(U \cup V \cup \{f\}, u))$, $q(h(U_{\text{first}}, f))$ and $q(h(U_{\text{first}} \cup \{f\}, u))$) for every $(f, u) \in (U \cup V \cup S)^2 \setminus U^2$, their computation is always performed incrementally. An alternative strategy would be to initialize a counter when required. There are, however, exponentially many calls where a given counter is useful. As a consequence, the cost of the on-demand initialization of the counter (scan of part of the tensor) multiplied by this number of calls exceeds the cost of maintaining the counter updated until used or useless. Thus, all counters are initialized while storing the dataset and, whenever elements are moved or removed from V, the counters are updated by only traversing the symmetric differences of the subspaces of T they cover. These updates can be further improved by replacing the counters of the types $q(h(U_{\text{first}} \cup \{f\}, u))$ and $q(h(U \cup V \cup \{f\}, u))$ by $q(h(h(U_{\text{first}}, f), u))$ and $q(h(h((U \cup V, f), u))$, i. e., by quantities of noise in the lower-bound and the upper-bound of the search space projected on pairs of elements in different dimensions. The desired quantities $q(h(U_{\text{first}} \cup \{f\}, u))$ and $q(h(U \cup V \cup \{f\}, u))$ can be computed, still without any access to T:

- $q(h(U_{\text{first}} \cup \{f\}, u)) = q(h(U_{\text{first}}, u)) + q(h(h(U_{\text{first}}, f), u))$;
- $q(h(U \cup V \cup \{f\}, u)) = q(h(U \cup V, u)) + q(h(h(U \cup V, f), u))$.

The two new types of counter involve much smaller subspaces (one dimension less) than the ones they replace. Thus, they do not need to be updated as often, hence the additional time gain.

C. Time Gain and Space Complexity

Without the counters presented in Section IV-B, verifying Definition 5 would require reading every value T_t that relates to an n-tuple $t \in \cup_{i=1}^{n} (U_i \cup V_i \cup S_i) \times (\prod_{j \neq i} U_j \cup V_j)$. To verify Definition 3, a subset of those n-tuples, $\prod_{i=1}^{n} U_i$, would be accessed. When updating the noise counters, `multidupehack` reads the values in T that relate to the n-tuples in $\cup_{i \neq d} (U_i \cup V_i \cup S_i) \times (\times_{j \notin \{d,i\}} U_j \cup V_j)$, where d is the dimension where the enumerated element is chosen. Thus, when mining an n-dimensional dataset, `multidupehack` is, roughly speaking, as fast as the naive approach running on a tensor with $n-1$ dimensions and the same numbers of elements per dimension. Notice however that this statement only holds when the time spent reading the counters, to verify Definitions 3 and 5, is not greater than the time spent updating them.

To verify Definitions 3 and 5, `multidupehack` respectively reads $|V| + 2 \sum_{i=1..n} |V_i| \sum_{j \neq i} |U_j|$ and $|S| + 2 \sum_{i=1..n} |S_i| \sum_{j \neq i} |U_j \cup V_j|$ counters. Overall, the time spent is $O(\sum_{i=1..n} |S_i| \sum_{j \neq i} |U_j \cup V_j|)$. This number is now compared to the time spent updating the counters, $O(\sum_{i \neq d} |U_i \cup V_i \cup S_i| \prod_{j \notin \{d,i\}} |U_j \cup V_j|)$. When $n = 2$, the time spent reading the counters is dominant. That cost is, however, below that of the naive approach (with no counter). When $n \geq 3$, the comparison depends on the actual cardinalities of the sets U_i, V_i and S_i ($i = 1..n$). Let us consider the worst case scenario where every element absent from $U \cup V$ is in S (i. e., $\forall i = 1..n$, $U_i \cup V_i \cup S_i = D_i$), a perfectly cubic dataset (i. e., $\exists D \in \mathbb{N} \mid \forall i = 1..n$, $|D_i| = D$) and a same number of elements in every S_i (i. e., $\exists S \in \mathbb{N} \mid \forall i = 1..n$, $|S_i| = S$). Under those assumptions, the costs become:

- $O((n-1)D(D-S)^{n-2})$ to update the counters;
- $O(nS(n-1)(D-S))$ to verify Definitions 3 and 5.

When $n = 3$, the costs of reading to the counters and updating them are on the same order, hence the gain stated in the first paragraph of this section. When $n \geq 4$, the gain is preserved since `multidupehack` spends more time updating the counters than reading them.

The sets U, V and S and all the counters associated with the elements they contain need to be copied at every first recursive call of `multidupehack`. When $n \in \{2, 3\}$, the counters occupy most of the memory. In this case, and since the depth of the recursion reaches, in the worst case, $\sum_{i=1}^{n} |D_i|$, the space complexity of `multidupehack` is $O((\sum_{i=1}^{n} |D_i|) \times (2\sum_{i=1}^{n} |D_i| + 2\sum_{i \neq j} |D_i \times D_j|)) = O(|D_i|^2 \times |D_j|)$, where D_i is the largest dimension and D_j the second largest. When $n \geq 4$, the space to store the data in a trie, $O(\prod_{i=1}^{n} |D_i|)$, usually dominates that of the counters, which remains unchanged. Because the second recursive call of `multidupehack` is a tail call, the variables (U, V, S and the counters) of the parent call can be safely reused. Not copying these variables reduces both the time and space requirements in a significant way.

D. Closedness "Jump"

Taking advantage of the counters of the type $q(h(U \cup V, f))$, `multidupehack` can cheaply check whether $U \cup V$ is connected. In that event, there is no need to pursue the enumeration as usual. Indeed, by Definition 4, none of the patterns strictly smaller than $U \cup V$ is closed. To avoid the useless enumeration of those patterns, `multidupehack` "jumps" to $U \cup V$, i. e., every call of `multidupehack` actually starts with the execution of the pseudo-code in Algorithm 2.

Algorithm 2 Closedness jump.

```
if U ∪ V is connected then
    U ← U ∪ V
    V ← (∅, ..., ∅)
end if
```

E. Choosing the Element to Enumerate

At every recursive call, any element $e \in V$ can be enumerated (function Choose in Algorithm 1). Nevertheless, respecting the so-called *fail-first principle* allows to reduce the size of the enumeration tree that is traversed, i. e., e should be chosen so that the search spaces after the two recursive calls are as small as possible. Since the search space after the second recursive call always is $V \setminus \{e\}$, the objective is the heuristic minimization of $|\{v \in V \setminus \{e\} \,|\, U \cup \{e\} \cup \{v\} \text{ is connected}\}|$. `multidupehack` first chooses the dimension in which e is taken; then it chooses e itself. The heuristic to choose the dimension is that of DATA-PEELER [10], i. e., the index d of the dimension, in which the element is taken, maximizes the following expression:

$$\sum_{i \neq d} |V_i| \prod_{j \notin \{d,i\}} |U_j| \ .$$

Given this index d, `multidupehack`'s choice of the enumerated element in V_d is "finer" than that of DATA-PEELER. Indeed, `multidupehack` has more information at its disposal: the noise counters listed in Section IV-B. It chooses the element that introduces as much noise as possible in every pattern that is recursively enumerated from the second call of `multidupehack` in Algorithm 1, i. e., it enumerates the element $e \in V_d$ that maximizes the counter $q(h(U_{\text{first}}, e))$.

The choice of the element to enumerate has a huge influence on the number of considered patterns, hence on the running time. Other sensible heuristics were tested. The one defined above has provided the best performance. Some one-step procedures that directly take advantage of the noise counters (e. g., the choice of the element $e \in V_i$ maximizing $\sum_{v \in V_{j \neq i}} q(h(h(U_{\text{first}}, e), v))$) led to slightly less recursive calls of Algorithm 1. However, the computational cost of such procedures ($O(|V|^2)$ in the previous example) exceed the enabled gains.

F. Piecewise (Anti)-Monotonicity

By reusing the enumeration principles of DATA-PEELER, `multidupehack` profits from the same large class of additional constraints: the piecewise (anti)-monotone ones. This class includes (but is not restricted to) any Boolean expression of monotone and anti-monotone constraints. In Algorithm 1, $\mathcal{C}$ is such a constraint that allows to both focus on the relevant patterns and fasten their extraction by search-space pruning. A constraint formally is a function taking a pattern at input and returning either true (the constraint is satisfied) or false (the constraint is violated). Here are the definitions of two constraints the experiments in Section V-B2 enforce to efficiently mine relevant patterns in dynamic graphs:

Definition 6: Given a set of timestamps $D_T \subseteq \mathbb{R}$, $n-1$ other dimensions of analysis $(D_i)_{i=2..n}$ and a threshold $\tau \in \mathbb{R}_+$, a pattern $(T, X) \in 2^{D_T} \times (\prod_{i=2}^{n} 2^{D_i})$ is *τ-contiguous* if and only if $\forall t \in [\min(T), \max(T)], \exists t' \in T$ s.t. $|t' - t| \leq \tau$.

Definition 7: Given a set of vertices D_N and $n-2$ other dimensions of analysis $(D_i)_{i=3..n}$, a pattern $(N_d, N_a, X) \in 2^{D_N} \times 2^{D_N} \times (\prod_{i=3}^{n} 2^{D_i})$ is *symmetric* if and only if $N_d = N_a$.

The definition of piecewise (anti)-monotonicity relies on the notion of (anti)-monotonicity per argument:

Definition 8: Given n dimensions of analysis $(D_i)_{i=1..n}$ and $i = 1..n$, a constraint $\mathcal{C}$ is *(anti)-monotone* w.r.t. the i^{th} argument if and only if $\forall (X_1, \ldots, X_n) \in \prod_{i=1..n} 2^{D_i}$,

$$\begin{cases} \text{(monotonicity) } \forall X_i' \in D_i, X_i \subseteq X_i' \Rightarrow \\ \Big(\mathcal{C}(X_1, \ldots, X_n) \Rightarrow \mathcal{C}(X_1, \ldots, X_{i-1}, X_i', X_{i+1}, \ldots, X_n)\Big) \\ \text{or} \\ \text{(anti-monotonicity) } \forall X_i' \in D_i, X_i \subseteq X_i' \Rightarrow \\ \Big(\mathcal{C}(X_1, \ldots, X_{i-1}, X_i', X_{i+1}, \ldots, X_n) \Rightarrow \mathcal{C}(X_1, \ldots, X_n)\Big) \end{cases}$$

Stated in English, a constraint is (anti)-monotone w.r.t. the i^{i} argument if and only if a pattern satisfying the constraint keeps on satisfying it either if its i^{th} argument is enlarged or if it is shrunk. A piecewise (anti)-monotone constraint is, by definition, (anti)-monotone w.r.t. each *occurrence* of each of its arguments:

Definition 9: A constraint is *piecewise (anti)-monotone* if and only if rewriting it by attributing a separate argument to every occurrence of its variables provides a new constraint that is (anti)-monotone w.r.t. each of its arguments.

For example, the τ-contiguity constraint (Definition 6) is piecewise (anti)-monotone. Indeed, rewriting it with one separate argument per occurrence of the only variable in its expression provides:

$$(T_1, T_2, T_3) \mapsto \forall t \in [\min(T_1), \max(T_2)], \exists t' \in T_3 \text{ s.t. } |t' - t| \leq \tau \ .$$

This constraint is (anti)-monotone w.r.t. each of its arguments because, assuming it is satisfied, it keeps on being satisfied when:

- T_1 is shrunk (anti-monotonicity w.r.t. the first argument);
- T_2 is shrunk (anti-monotonicity w.r.t. the second argument);
- T_3 is enlarged (monotonicity w.r.t. the third argument).

The symmetry constraint (see Definition 7) is piecewise (anti)-monotone too. To prove it, the equality must be turned into a conjunction of inclusions. Then, rewriting it with one separate argument per occurrence of the two variables provides:

$$(N_{d,1}, N_{d,2}, N_{a,1}, N_{a,2}) \mapsto N_{d,1} \subseteq N_{a,1} \wedge N_{a,2} \subseteq N_{d,2} \ .$$

This constraint is (anti)-monotone w.r.t. each of its arguments because, assuming it is satisfied, it keeps on being satisfied when:

- $N_{d,1}$ is shrunk (anti-monotonicity w.r.t. the first argument);
- $N_{d,2}$ is enlarged (monotonicity w.r.t. the second argument);
- $N_{a,1}$ is enlarged (monotonicity w.r.t. the third argument);
- $N_{a,2}$ is shrunk (anti-monotonicity w.r.t. the fourth argument).

At every recursive call of `multidupehack`, the monotone occurrences of a variable X_i are replaced by $U_i \cup V_i$, i. e., the largest i^{th} dimension of all patterns that may be recursively enumerated. On the contrary the anti-monotone occurrences of a variable X_i are replaced by U_i, i. e., the smallest i^{th} dimension of all patterns that may be recursively enumerated. By Definition 9, this instantiation violates the rewritten constraint only if none of the patterns, that would be recursively enumerated, satisfies the original constraints. That is why, in this case, `multidupehack` safely aborts the recursion.

The class of piecewise (anti)-monotone constraints is very large. It includes (but is not restricted to) any Boolean expression of monotone and anti-monotone constraints. This great expressive power makes `multidupehack` efficient in the numerous specific contexts where the relevant patterns can be specified via piecewise (anti)-monotone constraints. For example, the τ-contiguity and the symmetry constraints allow the efficient discovery of maximal cross-graph quasi-cliques in a set of timestamped graphs whose edges can be directed, weighted and labeled (see Section V-B2). We believe that this ability to efficiently discover specific patterns in any dataset that can be represented as a numeric tensor is a significant step toward the implementation of an inductive querying system [27] for itemset-like patterns.

V. Experimental Study

Both `multidupehack` and DCE are free software[3] implemented in C++ and compiled by GCC 4.4.3 with the O3 optimizations. This section reports experiments performed on a GNU/Linux™ system running on top of 1.2 GHz cores (but `multidupehack` and DCE are monothreaded) and 2 GB of RAM. DCE's ability to rank the patterns is disabled.

A. Comparison with DCE on Synthetic Data

1) Uncertain Tensor Generation: To quantitatively assess the quality of the closed patterns defined by both Definitions 3 and 4, "perfect" patterns (i. e., covering only '1' values) are randomly placed (possibly overlapping) in a null tensor. All values are then altered by some well-controlled noise. Finally, the tuples covered by the closed patterns in the resulting uncertain tensor are compared with those having '1' values before the application of noise.[4] Section VI justifies the choice of a tuple-based (rather than pattern-based) quality measure.

Definition 10: Given H, the set of tuples covered by the hidden patterns, and E, the set of tuples covered by the extracted patterns, the quality of the extraction is $\frac{|H \cap E|}{|H \cup E|}$.

"Perfect" tensors with two, three and four dimensions were generated. They all are "cubic", i. e., have the same number of elements in every dimension. To ease the interpretation, the number of values in the tensors (4096), the number elements per dimension of the hidden patterns (4) and the total number of tuples covered by these patterns (512 minus the overlapping) are kept constant in all settings. For instance, the 2-dimensional datasets are 64×64 matrices containing 32 4×4 hidden patterns. Since the patterns are "cubic", a given extraction of `multidupehack` or DCE is always parametrized with a same noise tolerance threshold ϵ for every dimension.

How to apply noise is, by itself, an interesting topic. The method proposed in this section improves what has been done in the data mining literature so far (switching, with a given probability, the presence/absence of a tuple). Consider that every '0'/'1' value in a "perfect" tensor stands for the empirical observation of the violation/satisfaction of the predicate the tensor encodes. If this observation is subject to uncertainty, a Bernoulli distribution of parameter p could describe the probability of a positive observation (i. e., of the satisfaction of the predicate) and p actually quantifies this uncertainty. As a consequence, p could be considered a noisy version of the value in the "perfect" tensor. After $\alpha - 1$ positive

[3] `multidupehack` is available, under the terms of the GNU GPLv3, at http://dcc.ufmg.br/~lcerf/en/prototypes.html#multidupehack.

[4] The commands generating the fuzzy tensors are available, under the terms of the GNU GPLv3, at http://dcc.ufmg.br/~lcerf/en/utilities.html#noisy

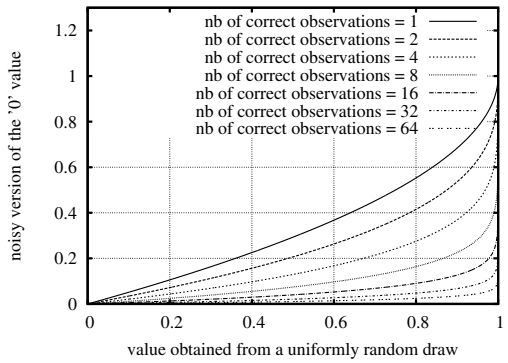

Fig. 4. To alter a value at '0' in a "perfect" tensor, a uniformly random number in $[0, 1]$ is drawn and the noisy value is read on the curve providing the desired level of noise (more "correct observations" mean less noise).

observations and $\beta - 1$ negative ones, the posterior distribution of p is given by the beta distribution with the parameters α and β. Therefore, the inverse of this beta distribution at a uniformly random value between 0 and 1 is a possible p that can be considered a noisy version of the value in the "perfect" tensor. In this process, α and β control the level of noise. We chose to only assume correct observations, i. e., to set $\alpha = 1$ (resp. $\beta = 1$) when applying noise on a '0' (resp. '1') value. The more correct observations, the less noise in the resulting tensors. Figure 4 plots the considered inverse beta distributions when altering a '0' value. We believe that the more realistic contexts are those obtained with at most 4 correct observations. However, considering "cleaner" data is interesting to experimentally understand the differences between `multidupehack`'s and DCE's definitions of a pattern.

2) Results: Both `multidupehack` and DCE perform faster under minimal size constraints. By querying the closed patterns with at least four elements per dimension (i. e., the exact "geometry" of the hidden patterns) and tolerating a lot of noise, `multidupehack` can exactly recover the hidden patterns. For instance, in the 4-dimensional tensors with four correct observations per tuple, `multidupehack` always exactly discovers those patterns with $\epsilon = 16$. Their extraction takes about 0.28 second (average over 50 randomly generated tensors). However, in real contexts, (1) the tensors contain patterns of various sizes, (2) the analyst does not know the sizes of the hidden patterns and (3) tolerating the whole noise covered by a large pattern usually is intractable. For those reasons, in the following, every extraction is performed under minimal size constraints of two elements per dimension.

Figures 5, 6 and 7 respectively plot the quality (see Definition 10), the number of extracted patterns and the running times obtained with both `multidupehack` and DCE. Those results are averages over 50 randomly generated tensors. The chosen noise tolerance parameters, ϵ, for both algorithms are comparable: the relative ones (for DCE) are the proportions of noise tolerated by `multidupehack` in the smallest extracted patterns (two elements per dimension), which are the most numerous. The quality results obtained with a per-element tolerance to noise (`multidupehack` or DCE with the post-process selecting the patterns satisfying such a definition) are significantly better than those with a global tolerance to noise (DCE without the post-process). With a global tolerance to noise, the quality actually decreases when the tensor becomes very "clean" (many correct observations per tuple). Indeed, some extracted patterns partially cover the almost unaltered hidden patterns but go beyond them too (see Figure 3). Tolerating too much noise is harmful with both definitions.

Even with the filtering post-process, DCE extracts far more patterns than `multidupehack`, especially when "clean" tensors are mined. The reason is given in Section III-B: DCE does not (and theoretically cannot) enforce a closedness constraint, i. e., DCE outputs every "sub-pattern" of a valid pattern as long as the minimal size constraints are satisfied. In the "cleanest" settings (32 or 64 correct observations), `multidupehack` usually exactly recovers the hidden patterns.

The most significant advantage of `multidupehack` over DCE is its fastness. In those experiments, `multidupehack` almost always runs in less than 10 seconds. In contrast, some curves are missing in Figures 5, 6 and 7 because DCE could not terminate in a reasonable time. For instance, with $\epsilon = 0.25$, DCE was aborted after twelve hours of computation on a 2-dimensional dataset with one correct observation per tuple. On the same tensor, with the analog absolute noise tolerance ($\epsilon = 0.5$), `multidupehack` returns the 1688 closed patterns in 0.27 second. Without the post-process, DCE's running times are slightly worse because many more patterns are output.

B. Discovering Real-Life Closed Patterns

1) Influential Groups in Twitter: The micro-blogging service Twitter is particularly popular in Brazil. Tweets about the Brazilian soccer championship were collected from the 45^{th} week of 2010 to the 6^{th} week of 2011 (14 weeks) and classified w.r.t. to the mentioned team(s) (supervised classification method, which is out of the scope of this paper). How many times a user is retweeted (i. e., other users "repeat" her tweets) is known to be a good measure of her influence [1]. 126,566 users were retweeted at least once during the considered period. A 3-dimensional tensor gives how many times each of them is retweeted (over all her messages) during a given week when writing about a given soccer team (among 20). Those data are turned fuzzy via a logistic function with a growth rate of 0.2 and centered on 20 retweets (Figure 2 plots it). In this way, the users retweeted twenty times, during a given week when talking about a given team, are considered, in this context, "moderately influential" (0.5 in the uncertain tensor). The closed patterns discovered in this tensor always involve the most famous teams. For instance many patterns stand for influential supporters of Corinthians and São Paulo talking about those two famous (and rival) teams during the week of the game opposing them and several weeks after. Such patterns are discovered for Flamengo and Vasco da Gama too.

To discover closed patterns involving less popular teams, the influence of a user during a given week must be quantified w.r.t. the popularity of the team. To do so the retweet tensor is

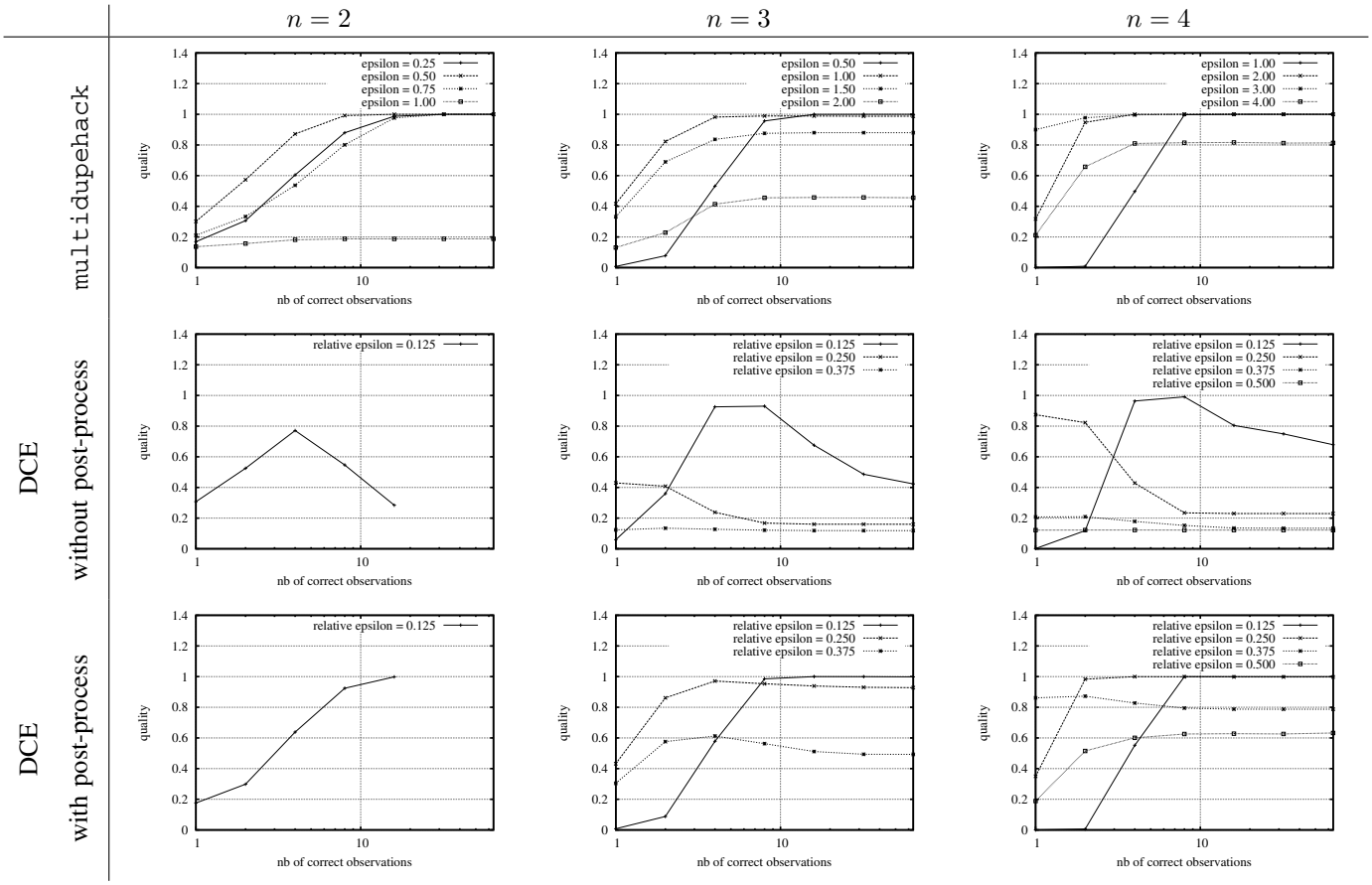

Fig. 5. Quality results of `multidupehack` and DCE on synthetic tensors.

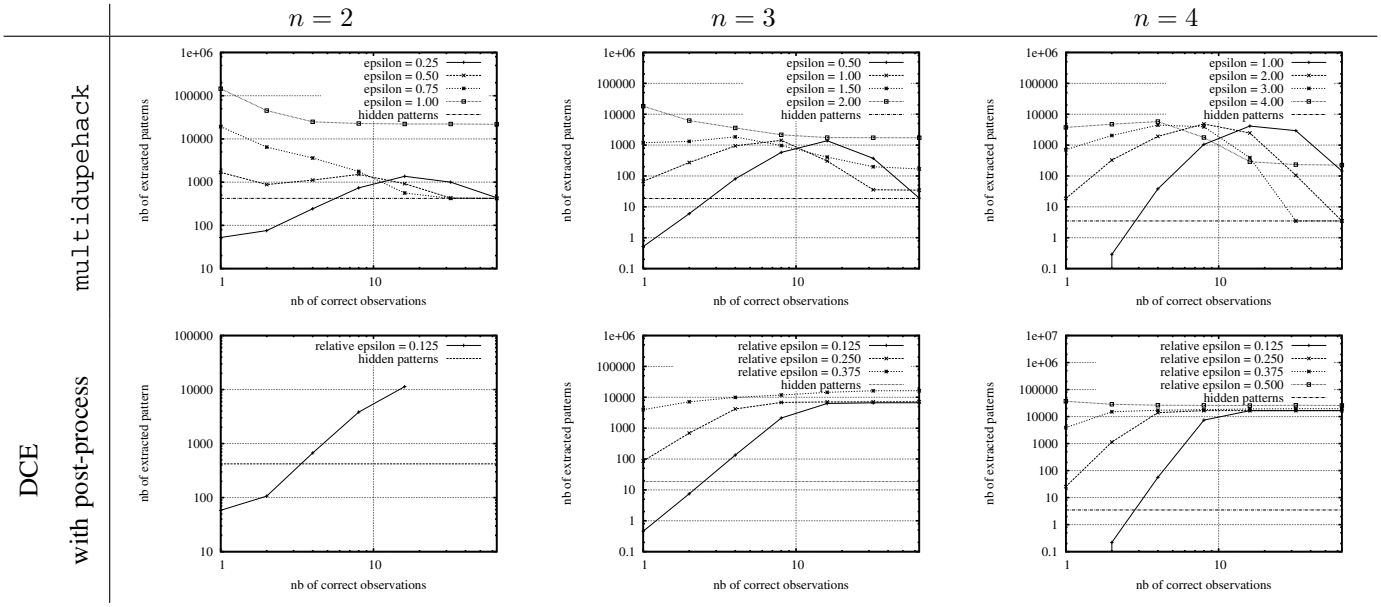

Fig. 6. Number of patterns extracted by `multidupehack` and DCE in synthetic tensors.

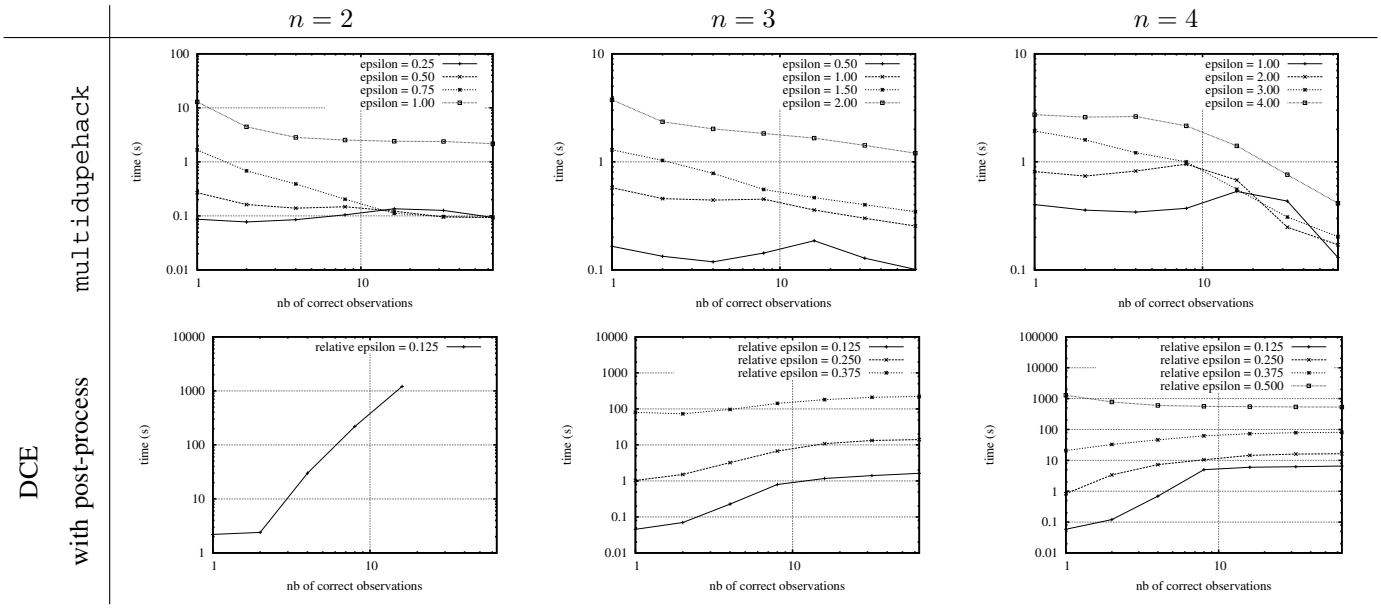

Fig. 7. Running time of `multidupehack` and DCE on synthetic tensors.

normalized so that the sum of the values associated with any team becomes the average number of retweets per team. The logistic function described above is then reused to turn the data fuzzy. The resulting $14 \times 20 \times 126,566$ fuzzy tensor can be said "sparse": the sum of all its membership degrees only is $14,042$, i. e., $\frac{14,042}{14 \times 20 \times 126,566} \approx 0.0004$ times the maximal possible value (every membership degree at 1). The closed patterns with at least three weeks, four teams and two users are extracted in that tensor. With $\epsilon_{\text{week}} = 3$, $\epsilon_{\text{team}} = 2$ and $\epsilon_{\text{user}} = 4$, `multidupehack` discovers 42 such patterns in 56 seconds. Because of the strong constraint on the number of teams, these closed patterns involve famous soccer commentators rather than supporters of one particular team. Here is one of them:

$$(\{\text{week 45 of 2010, week 47 of 2010, week 48 of 2010}\},\\ \{\text{Avaí, Fluminense, Goiás, Santos, São Paulo}\},\\ \{\texttt{globoesportecom}, \texttt{milton\_neves}\})$$

It means that both commentators from the Globo and the Bandeirantes TV channels were influential during the last moments of the championship (which ended during the 48$^{\text{th}}$ week of 2010) when writing about the team that was leading it (Fluminense), those that might be relegated to the "Série B" (Avaí and Goiás) and São Paulo, which was fighting for a position in the American "Copa Libertadores". The presence of Santos may be due to the rise in popularity its young and promising player Neymar. As expected, none of the three weeks during which no match was played is present in any discovered pattern.

We tested whether DCE could extract the "cleanest" pattern (hence the easiest to extract) among the 42 discovered by `multidupehack`. This pattern globally covers slightly more than 11.3% of noise. Thus, DCE has been parametrized with a 11.4% tolerance threshold and the same size constraints as above (at least three weeks, four teams and two users). After 100 hours of (unterminated) computation, DCE had not output any pattern yet. In contrast, it takes `multidupehack` less than 16 seconds to discover this pattern with the parametrization that barely allows its extraction ($(\epsilon_{\text{week}}, \epsilon_{\text{team}}, \epsilon_{\text{user}}) = (1.44, 1.37, 1.63)$). This shows that `multidupehack` opens up applicative perspectives that are out of reach of DCE for performance reasons.

2) Usage Patterns in a Transportation Network: As proved in Section IV-F, both the τ-contiguity and the symmetry constraints (see Definitions 6 and 7) are piecewise (anti)-monotone. That means `multidupehack` can efficiently enforce them. Given a dynamic graph (i. e., a set of timestamped graphs), the closed patterns satisfying the symmetry constraint between the two dimensions of the graph adjacency matrices are some kind of maximal cross-graph quasi-cliques. The involved definition is, however, less constrained than that of [13], [14] (see Section II). In addition, the τ-contiguity constraint forces the timestamps of the graphs involved in a closed pattern to be separated from each other by, at most, a τ-long period. Moreover, because `multidupehack` can mine any numeric tensor, the edges of the dynamic graph can be directed, weighted and labeled (the set of labels constitutes an additional dimension of analysis).

All these characteristics are necessary to a good usage analysis of the public bicycle network in Lyon, France. This network consists in 327 stations (the vertices of the graph) where bicycles can be rented and returned. The behaviors of the users clearly evolve along the day (24 1-hour periods that constitute the temporal dimension of the dynamic graph) and depend on the day of the week (7 edge labels constituting a fourth dimension of analysis). A two-year log of this system defines the $327 \times 327 \times 24 \times 7$ tensor. Every value in it is a number of rides (in the log) from a departure station (where the bicycle is rented) to an arrival station (where the bicycle

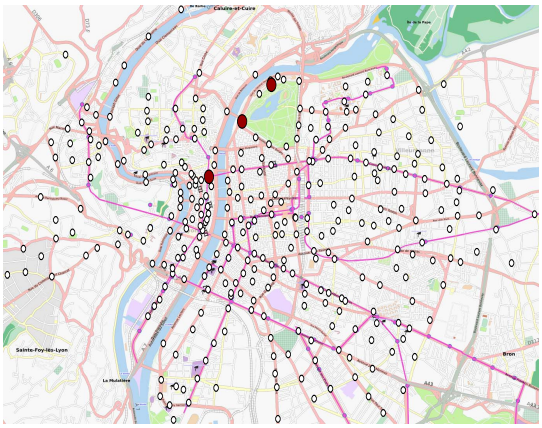

Fig. 8. Geographic positions of three stations, in the main park of Lyon and on the Rhône river side, that exchange many bicycles during the week-end between 2pm and 8pm.

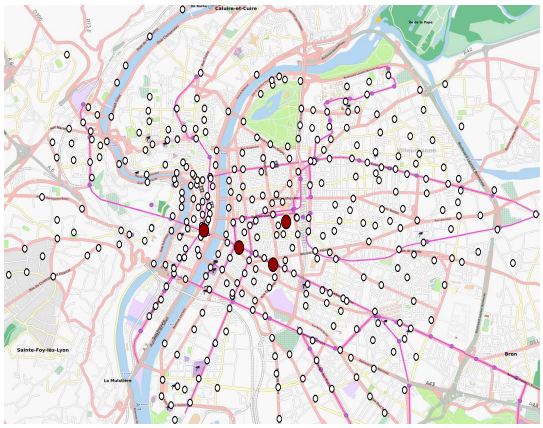

Fig. 9. Geographic positions of four stations, in the working and commercial districts of Lyon, that exchange many bicycles on Mondays, Tuesdays and Wednesdays between 4pm and 7pm.

is returned) during a day of the week and a 1-hour period. Of course, inverting the departure and the arrival leads to a value that can be very different, i. e., the graph is directed. The tensor is normalized so that every day of the week and every 1-hour period have a same total weight (the usage variations due to them are already known). Finally, a logistic function turns the tensor fuzzy. That fuzzy tensor is about eight times denser than the one mined in the previous section: the sum of all the membership degrees is $83,140$, i. e., $\frac{83,140}{327\times327\times24\times7} \approx 0,0031$ times the maximal possible value.

In this tougher context, `multidupehack` remains tractable thanks to the use of constraints. Indeed, the closed patterns with at least three identical departure and arrival stations (symmetry constraint), two days of week and three 3-hour-contiguous time periods are queried. `multidupehack` takes 3 minutes and 50 seconds to list the 656 such patterns with $\epsilon_{\text{day}} = 2.7$ and $\epsilon_{\text{departure}} = \epsilon_{\text{arrival}} = \epsilon_{\text{hour}} = 1.8$. Given our background knowledge of the city, these patterns make sense. Those involving Saturday and Sunday afternoons stand for leisure trips either in the historical center, along the Rhône river or in the main park of Lyon (see Figure 8), whereas those happening during the working days are concentrated in the working and the commercial areas of the city and usually occur in the evenings (see Figure 9). Increasing the noise tolerance to $\epsilon_{\text{day}} = 4.5$ and $\epsilon_{\text{departure}} = \epsilon_{\text{arrival}} = \epsilon_{\text{hour}} = 3$ allows to discover more and larger patterns: 2804 closed patterns are returned within 16 minutes and 21 seconds.

VI. Discussion

This article both theoretically and empirically argues for an effective an efficient way to tolerate noise in local patterns. Somewhat extending results in [22], which only considers 2-dimensional Boolean matrices, it has explained why the noise should be upper-bounded *per element* rather than per tuple (loss of information) or per pattern (extraction of pattern going beyond the "real" ones). The experimental results in Figure 5 clearly validate this choice: the extracted patterns cover the tuples that ought to be covered. Unfortunately, the completeness of the approach (a closed pattern is extracted *if and only if* it satisfies the definition) practically prevents the analyst from setting noise tolerance thresholds that are large enough for every hidden pattern to be recovered in its entirety by one single closed pattern. Indeed, running times quickly increase with these thresholds (see Figure 7) and, in practice, the extracted pieces of knowledge are overlapping fragments (though larger hence better fragments than without any noise tolerance) tiling the "real" patterns the analyst would like to discover.

To actually reach those "real" patterns, the completeness demand has to be revised downwards. Nevertheless, our proposal remains valuable to attain this goal. Indeed, heuristic methods, such as [28], do not directly search the patterns from the data. Instead, they cluster fragments of these patterns (what justifies the tuple-based quality measure chosen in Section V-A). By using a *complete* collection of such fragments, the lossy heuristics are delayed as far as possible in the knowledge discovery process, which therefore becomes more trustworthy.

To be able to post-process the fragments, their number must remain reasonable (e. g., the time complexity of their hierarchical agglomeration, proposed in [28], is quadratic). This is one of the reasons why an *absolute* tolerance to noise is preferable to a relative one. Indeed, the downward closure (Theorem 1) only holds in that case and is the key to enforce a closedness constraint that restricts the collection of patterns to the most informative ones (see [25]) without any loss of information. Figure 6 demonstrates its efficiency. Nevertheless, the most significant reason for the choice of an absolute tolerance to noise is the ability to extract the patterns faster. Indeed, and again thanks to the downward closure, subspaces of the search space are pruned earlier because the algorithm does not care whether the current absolute noise might be a low proportion of larger patterns. By reusing the efficient enumeration principles of Data-Peeler [10], `multidupehack` is orders of magnitude faster than DCE [2], [3] that tolerates noise in a relative way (see Figure 7). It could be argued that the reason for this significant performance

gain is also to be found in the more constrained definition of the patterns (noise tolerated "per element" vs. "per pattern"). However no enumeration principle is currently known to efficiently list patterns tolerating noise per element in a relative way. DCE enforces this better definition as a post-process and, in the restricted context of Boolean matrices, [29] directly extracts such patterns but the obtained running times are orders of magnitude worse than the same authors' previous proposal with an absolute tolerance to noise [7].

Reusing DATA-PEELER's enumeration principles also allows to efficiently enforce expressive constraints (any piecewise (anti)-monotone constraint). This has a major impact on both the quality of the discovered patterns (the analyst can focus on those satisfying some relevant properties) and the time to list them (the search space being pruned w.r.t. these properties). For instance, Section V-B2 has reported the tractable extraction of 3-hour-contiguous quasi-cliques in a real-life weighted directed network evolving in two temporal dimensions. Without the symmetry (to focus on *cliques*) and the τ-contiguity constraints, it would be impossible to discover every noise tolerant pattern in this large and dense dataset.

Last, but not least, the complete extraction, in numerical data (no loss of information inherent to binning), of "itemset-like" patterns with their supporting sets is a rather novel problem. Only DCE and `multidupehack` have tackled it so far and directly in the context of n-dimensional data. Many real-life data being both numerical and multidimensional (say objects described by numerical attributes at different timestamps and/or in different contexts), such developments open up many new applicative perspectives.

VII. CONCLUSION

Until recently, the discovery of local patterns in numerical data was either requiring a binning step or relying on lossy heuristics. In contrast, `multidupehack` does not imply any loss of information before or during the pattern extraction. It processes uncertain tensors of any dimensionality, i. e., collections of n-tuples associated with degrees of satisfaction of the studied predicates. This paper has theoretically and empirically demonstrated that the chosen definition of the underlying noise tolerance matches high-quality patterns and that `multidupehack` is orders of magnitude faster than its only competitor. Expressive constraints of relevance can be additionally enforced so that more meaningful closed patterns are discovered in larger tensors. For instance, it has been shown that cross-graph quasi-cliques can be discovered in a weighted directed graph evolving in two temporal dimensions.

VIII. ACKNOWLEDGMENTS

This research has been partially funded by the CNPq, FAPEMIG and InWeb.

REFERENCES

[1] H. Kwak, C. Lee, H. Park, and S. Moon, "What is twitter, a social network or a news media?" in *Proc. WWW'10*, Apr. 2010, pp. 591–600.

[2] E. Georgii, K. Tsuda, and B. Schölkopf, "Multi-way set enumeration in weight tensors," *Machine Learning*, vol. 82, no. 2, pp. 123–155, Feb. 2011.

[3] ——, "Multi-way set enumeration in real-valued tensors," in *Proc. DMMT'09*, Jun. 2009, pp. 32–41.

[4] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, "Efficient mining of association rules using closed itemset lattices," *Information Systems*, vol. 24, no. 1, pp. 25–46, Mar. 1999.

[5] M. J. Zaki and C.-J. Hsiao, "CHARM: An efficient algorithm for closed association rule mining," Computer Science Department, Rensselaer Polytechnic Institute, Tech. Rep. 99-10, 1999.

[6] R. Gupta, G. Fang, B. Field, M. Steinbach, and V. Kumar, "Quantitative evaluation of approximate frequent pattern mining algorithms," in *Proc. KDD'08*, Aug. 2008, pp. 301–309.

[7] A. K. Poernomo and V. Gopalkrishnan, "Efficient computation of partial-support for mining interesting itemsets," in *Proc. SDM'09*, May 2009, pp. 1014–1025.

[8] L. Ji, K.-L. Tan, and A. K. H. Tung, "Mining frequent closed cubes in 3D data sets," in *Proc. VLDB'06*, Sep. 2006, pp. 811–822.

[9] R. Jaschke, A. Hotho, C. Schmitz, B. Ganter, and G. Stumme, "TRIAS–an algorithm for mining iceberg tri-lattices," in *Proc. ICDM'06*, Dec. 2006, pp. 907–911.

[10] L. Cerf, J. Besson, C. Robardet, and J.-F. Boulicaut, "Closed patterns meet n-ary relations," *ACM Trans. on Knowledge Discovery from Data*, vol. 3, no. 1, pp. 3:1–3:36, Mar. 2009.

[11] L. Cerf, J. Besson, K.-N. T. Nguyen, and J.-F. Boulicaut, "Closed and noise-tolerant patterns in n-ary relations," *Data Mining and Knowledge Discovery*, vol. 26, no. 3, pp. 574–619, May 2013.

[12] K. Sim, G. Liu, V. Gopalkrishnan, and J. Li, "A case study on financial ratios via cross-graph quasi-bicliques," *Information Sciences*, vol. 181, no. 1, pp. 201–216, Jan. 2011.

[13] D. Jiang and J. Pei, "Mining frequent cross-graph quasi-cliques," *ACM Trans. on Knowledge Discovery from Data*, vol. 2, no. 4, pp. 16:1–16:42, Jan. 2009.

[14] Z. Zeng, J. Wang, L. Zhou, and G. Karypis, "Out-of-core coherent closed quasi-clique mining from large dense graph databases," *ACM Trans. on Database Systems*, vol. 32, no. 2, pp. 13–42, Jun. 2007.

[15] V. Ganti, J. Gehrke, and R. Ramakrishnan, "CACTUS—clustering categorical data using summaries," in *Proc. KDD'99*, Aug. 1999, pp. 73–83.

[16] M. J. Zaki, M. Peters, I. Assent, and T. Seidl, "CLICKS: An effective algorithm for mining subspace clusters in categorical datasets," *Data & Knowledge Engineering*, vol. 60, no. 1, pp. 51–70, Jan. 2007.

[17] L. Zhao and M. J. Zaki, "TRICLUSTER: An effective algorithm for mining coherent clusters in 3D microarray data," in *Proc. SIGMOD'05*, Jun. 2005, pp. 694–705.

[18] X. Xu, Y. Lu, K.-L. Tan, and A. K. H. Tung, "Finding time-lagged 3D clusters," in *Proc. ICDE'09*, Apr. 2009, pp. 445–456.

[19] C.-K. Chui, B. Kao, and E. Hung, "Mining frequent itemsets from uncertain data," in *Advances in Knowledge Discovery and Data Mining, Proc. PAKDD'07*, May 2007, pp. 47–58.

[20] T. Calders, C. Garboni, and B. Goethals, "Efficient pattern mining of uncertain data with sampling," in *Advances in Knowledge Discovery and Data Mining, Proc. PAKDD'10*, Jun. 2010, pp. 480–487.

[21] ——, "Approximation of frequentness probability of itemsets in uncertain data," in *Proc. ICDM'10*, Dec. 2010, pp. 749–754.

[22] J. Liu, S. Paulsen, X. Sun, W. Wang, A. B. Nobel, and J. Prins, "Mining approximate frequent itemsets in the presence of noise: Algorithm and analysis," in *Proc. SDM'06*, Apr. 2006, pp. 405–416.

[23] D. Avis and K. Fukuda, "Reverse search for enumeration," *Discrete Applied Mathematics*, vol. 65, no. 1–3, pp. 21–46, Mar. 1996.

[24] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *Proc. VLDB'94*, Sep. 1994, pp. 487–499.

[25] A. Gallo, A. Mammone, T. D. Bie, M. Turchi, and N. Cristianini, "From frequent itemsets to informative patterns," Univ. of Bristol, Tech. Rep. 123936, 2009.

[26] H. Mannila and H. Toivonen, "Multiple uses of frequent sets and condensed representations," in *Proc. KDD'96*, Aug. 1996, pp. 189–194.

[27] T. Imielinski and H. Mannila, "A database perspective on knowledge discovery," *Communications of the ACM*, vol. 39, no. 11, pp. 58–64, Nov. 1996.

[28] L. Cerf, P.-N. Mougel, and J.-F. Boulicaut, "Agglomerating local patterns hierarchically with ALPHA," in *Proc. CIKM'09*, Nov. 2009, pp. 1753–1756.

[29] A. K. Poernomo and V. Gopalkrishnan, "Towards efficient mining of proportional fault-tolerant frequent itemsets," in *Proc. KDD'09*, Jul. 2009, pp. 697–706.