Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 126 (2018) 403–412

Procedia Computer Science

www.elsevier.com/locate/procedia

International Conference on Knowledge Based and Intelligent Information and Engineering Systems, KES2018, 3-5 September 2018, Belgrade, Serbia

Mining High-Utility Patterns in Uncertain Tensors

Aurélien Coussat, Nicolas Nadisic, Loïc Cerf*

Department of Computer Science, Universidade Federal de Minas Gerais, Belo Horizonte, Brazil

Abstract

Transactional datasets are 0/1 matrices, which generically stand for objects having Boolean properties. If every cell of the matrix is additionally associated with a real number called *utility*, a *high-utility itemset* relates to a all-ones sub-matrix with utilities that sum to a high-enough value. This article shows that "having a total utility exceeding a threshold" is a piecewise (anti-)monotone constraint, even in presence of both positive and negative utilities. For that reason, a generic algorithm, `multidupehack`, can prune the search of the high-utility patterns defined in a broader context than the 0/1 matrix: the *uncertain tensor*. Moreover, the utilities may relate to only some of the dimensions, the patterns can be forced (or not) to be closed and to satisfy additional constraints. A real-world application, which exploits all those possibilities, is presented. Despite its versatility, the proposal is also shown competitive when it comes to mining 0/1 matrices, a special case treated by dozens of specific algorithms.

Keywords:
Pattern mining, Closed patterns, High-utility itemsets, Negative utility, Uncertain tensors

1. Introduction

In a 0/1 matrix, an itemset is a subset of columns and its supporting set is a subset of rows. By definition, they define together a sub-matrix that must be full of 1. For instance, given customers buying products, an analyst can see these data as a 0/1 matrix: each 1 indicates that the customer (in row) bought the product (in column). In those data, an itemset is a subset of products. Its supporting set is the subset of customers who bought all the products in the itemset. The pattern reveals correlations between the customers, who bought a same subset of products, and between the products, which were all bought by the same customers. Nevertheless, the analyst may have at its disposal additional information to more finely define the patterns she is interested in: when the sell was made (for a dynamic analysis of the purchase behaviors), what is the acquired quantity (to assess how relevant is the sell), how much the customer paid (to focus of patterns generating significant profits), etc. Table 1a presents a sample of such data.

A so-called *high-utility itemset* can take into account the `price` information in Table 1a. It is an itemset that is further constrained: the total amount of money that the customers (in the supporting set) spent on the products (in the itemset) must exceed a threshold, that the analyst fixes. That additional constraint filters out patterns with little impact

* Corresponding author. Tel.: +55-31-3409-7589.
E-mail address: lcerf@dcc.ufmg.br

10.1016/j.procs.2018.07.274

Table 1: Raw data, a 3-way uncertain tensor derived from them (quantities are mapped to values in [0, 1]) and the utility function.

(a) Supermarket sale data.

day	customer	product	quantity	price (€)
01	Alice	Yogurt	6	7
01	Alice	Egg	6	2
01	Bob	Egg	12	4
01	Bob	Wine	1	20
01	Bob	Water	1	1
02	Alice	Wine	2	40
02	Dave	Yogurt	6	5
⋮	⋮	⋮	⋮	⋮

(b) 3-way uncertain tensor.

day	customer	product	↦	T_t
01	Alice	Yogurt	↦	0.6
01	Alice	Egg	↦	0.5
01	Bob	Egg	↦	0.9
01	Bob	Wine	↦	0.8
01	Bob	Water	↦	0.3
02	Alice	Wine	↦	1
02	Dave	Yogurt	↦	0.6
⋮	⋮	⋮	⋮	⋮

(c) Utility function ($I = \{1, 2, 3\}$).

day	customer	product	↦	$u(t)$
01	Alice	Yogurt	↦	7
01	Alice	Egg	↦	2
01	Bob	Egg	↦	4
01	Bob	Wine	↦	20
01	Bob	Water	↦	1
02	Alice	Wine	↦	40
02	Dave	Yogurt	↦	5
⋮	⋮	⋮	⋮	⋮

on the turnover. Many algorithms handle it during the search of the patterns, i. e., the constraint prunes the pattern space and the high-utility itemsets can be discovered in large datasets where listing all itemsets (to then keep those of high-utility) is intractable.

This article studies a generalized version of the minimal-utility constraint: the utilities can be attached to tuples that need not involve elements in all the dimensions (e. g., every `product`, rather than every pair (`customer`, `product`), can have a `price`) and the utilities can be positive *and* negative. Even generalized, the constraint is shown to be *piecewise (anti-)monotone*, a mathematical property that allows some algorithms to prune the search of the high-utility patterns. `multidupehack`[1] is such an algorithm. Not only it efficiently handles any number of piecewise (anti-)monotone constraint, but it is not restricted to 0/1 matrices: it can mine *uncertain tensors*, i. e., tensors with values in [0, 1] that quantify to what extent the tuple is present in the dataset. Given Table 1a, the multidimensional aspect allows to take into consideration the `days` the purchases were made and seasonal behaviors can be discovered because a pattern becomes a subset of `products` bought by a subset of `customers` during a subset of `days`. The uncertain aspect allows to take into account the purchased `quantities`, which are turned into *membership degrees* in [0, 1], and not necessarily 0 or 1. In Table 1b, the analyst considered that buying six eggs is moderately significant (membership degree equal to 0.5), less significant than buying a dozen (membership degree equal to 0.9). `multidupehack` offers additional possibilities such as the search of *closed* patterns. This article proposes to implement the minimal-utility constraint in `multidupehack` to benefit from all those possibilities. That represents a significant advance w.r.t. the state-of-the-art algorithms, which individually tackle at most one of the generalizations listed above.

After some definitions in Section 2, Section 3 formalizes the general problem. Section 4 lists the state-of-the-art algorithms that solve special cases. Section 5 proves the piecewise (anti-)monotonicity of the minimal-utility constraint and explains how `multidupehack` enforces it. In Section 6, the proposal is shown competitive when applied to the special cases that existing algorithms handle. Section 6 reports as well experiments on a 3-way uncertain tensor, thereby demonstrating the usefulness of a general framework. Finally, Section 7 concludes the article.

2. Definitions

Given $n \in \mathbb{N}$ *dimensions* (i. e., n finite sets, assumed disjoint w.l.o.g.) $D_1, \ldots, D_n$, an *uncertain tensor* T, also known in the literature as an *n-ary fuzzy relation*, maps any n-tuple $t \in \prod_{i=1}^{n} D_i$ (where $\prod$ denotes the Cartesian product) to a value $T_t \in [0, 1]$, called *membership degree* of t. See Table 1b for an example of 3-way uncertain tensor. Uncertain tensors generalize both 0/1 tensors (with values in $\{0, 1\}$) and uncertain matrices (when $n = 2$).

A *pattern* is, in this article, n subsets of each of the n dimensions $D_1, \ldots, D_n$. Formally, $(X_1, \ldots, X_n)$ is a pattern if and only if $\forall i \in \{1, \ldots, n\}$, $X_i \subseteq D_i$. The pattern $(X_1, \ldots, X_n)$ *covers* the n-tuple $t \in \prod_{i=1}^{n} D_i$ if and only if $t \in \prod_{i=1}^{n} X_i$.

The above definitions are purely syntactical. To be semantically relevant, a pattern must mostly cover n-tuples having membership degrees close to 1. Cerf and Meira have argued, both theoretically and empirically, for the following semantics[1]: given n noise-tolerance thresholds $(\epsilon_1, \ldots, \epsilon_n) \in \mathbb{R}_+^n$, a pattern $(X_1, \ldots, X_n)$ is an ET-n-set (short for *Error-Tolerant n-set*) in T if and only if $\forall i \in \{1, \ldots, n\}$, $\forall x \in X_i$, $\sum_{t \in \prod_{i=1}^n X_i \text{ s.t. } t_i = x} 1 - T_t \leq \epsilon_i$, where t_i denotes the i^{th} component of the n-tuple t. In that definition of a semantically relevant pattern, $1 - T_t$ can be seen as an amount of noise to tolerate to have the pattern cover the n-tuple t. By definition, summing those amounts over the covered n-tuples with a fixed component x, involved in the ET-n-set, does not exceed a noise-tolerance threshold ϵ_i, which depends on the dimension x is taken in. The ET-n-sets generalize the patterns that only cover membership degrees equal to 1: $(\epsilon_1, \ldots, \epsilon_n) = (0, \ldots, 0)$ specifies these patterns, which tolerate no noise.

A pattern is *closed in the i^{th} dimension* if substituting its i^{th} set with any proper superset always produces a pattern that is not a closed ET-n-set. Formally, given $i \in \{1, \ldots, n\}$, a pattern $(X_1, \ldots, X_i, \ldots, X_n)$ is closed in the i^{th} dimension if and only if $\forall X_i' \supset X_i$, $(X_1, \ldots, X_i', \ldots, X_n)$ is not an ET-n-set. The classical itemsets (along with their supporting sets) in a 0/1 matrix therefore are the ET-2-sets, with $\epsilon_1 = \epsilon_2 = 0$, that are closed in the dimension of the supporting set. A *closed ET-n-set* is an ET-n-set that is closed in all n dimensions. The closed itemsets (along with their supporting sets) therefore are the closed ET-2-set with $\epsilon_1 = \epsilon_2 = 0$.

Given $I \subseteq \{1, \ldots, n\}$, a *utility function* u maps any $|I|$-tuple $t \in \prod_{i \in I} D_i$ with a (positive or negative) value $u(t) \in \mathbb{R}$, the *utility* of t. See Table 1c for an example of utility function. In that example, the utility is the `price`. It is associated with every individual purchase, identified by the `day`, the `customer` and the `product`, i. e., $I = \{1, 2, 3\}$. The *utility of a pattern* $(X_1, \ldots, X_n)$ is the sum of the utilities of the $|I|$-tuples in $\prod_{i \in I} X_i$:

$$\sum_{t \in \prod_{i \in I} X_i} u(t) \ .$$

A constraint is a function that takes a pattern at input and returns either "true" (the pattern satisfies the constraint) or "false" (the pattern violates the constraint). Given a threshold $\alpha \in \mathbb{R}$, the α-minimal-utility constraint, denoted $C_{\alpha\text{-min-utility}}$ and only satisfied by the so-called *high-utility patterns*, is:

$$(X_1, \ldots, X_n) \mapsto \sum_{t \in \prod_{i \in I} X_i} u(t) \geq \alpha \ .$$

3. Problem statement

Given an uncertain tensor $T \in [0, 1]^{\prod_{i=1}^n D_i}$, n noise tolerance thresholds $(\epsilon_1, \ldots, \epsilon_n) \in \mathbb{R}_+^n$, a subset $I \subseteq \{1, \ldots, n\}$, a utility function $u \in \mathbb{R}^{\prod_{i \in I} D_i}$ and a minimal pattern utility $\alpha \in \mathbb{R}$, this article tackles the correct and complete enumeration of the high-utility ET-n-sets. Additionally, the ET-n-sets can be required to be closed in any number of dimensions and other constraints on the patterns can be enforced.

4. Related work

Literally dozens of algorithms have been proposed to solve special cases of the problem in Section 3. All those works focus on the search of high-utility patterns in matrices, i. e., $n = 2$. The search in 0/1 matrices of high-utility patterns tolerating no noise (i. e., $\epsilon_1 = \epsilon_2 = 0$) is, by far, the most studied case. If the patterns are only forced to be closed in the dimension of the supporting set, and I only contains the index of the other dimension, the minimal-utility constraint is actually called *minimal-sum constraint*[2]. If I is, instead, $\{1, 2\}$, the returned patterns have been known under the name *high-utility itemsets* since Yao et al.'s seminal article[3]. The algorithm they propose only handles positive utilities. So do many subsequent algorithms. To the best of our knowledge, FHM[4], EFIM[5], ULB-Miner[6] and mHUIMiner[7] are, today, the fastest algorithms to list the high-utility itemsets in a 0/1 matrix when the utilities are all positive and no noise is tolerated. FHM+[8] additionally forces the returned itemsets to involve at least and at most user-defined numbers of columns (the *items*).

HUINIV-Mine[9], FHN[10] and GHUM[11] have been specifically designed to list high-utility itemsets (still without any tolerance to noise) in presence of negative utilities. Every algorithm cited so far forces the patterns to be closed in the

dimension of the supporting set but not in the other dimension. When every utility is positive, a pattern that is closed in all dimensions is necessarily of higher utility than any of its sub-patterns, which are also less informative for being smaller. That is why, given a 0/1 matrix and only positive utilities, CHUD[12], CHUI-Miner[13] and EFIM-Closed[14] only list the closed high-utility itemsets, still without any tolerance to noise. CHUI-Miner and EFIM-Closed are faster than CHUD. When some utilities are negative, mining high-utilities itemsets that are closed in the dimension of the supporting set may actually mean missing patterns that would be of high-utility if their supporting sets were restricted to the rows contributing positively to the utility of the pattern[9]. To the best of our knowledge, this articles proposes the first algorithm to possibly list high-utility patterns that need not be closed in any of the dimensions. It is as well the first algorithm that can mine high-utility patterns in tensors, and even uncertain tensors, of any dimensionality.

PHUI-List[15] and MUHUI[16] mine high-utility itemsets in 0/1 matrices whose rows have existing probabilities. HUPNU[17] considers probabilities attached to the cells of the matrix, i. e., an uncertain matrix. Contrary to PHUI-List and MUHUI, HUPNU handles negative utilities too. Those three algorithms constrain the expected number of rows (assumed independent) involved in a pattern to be above a user-defined threshold. The utility of a pattern is still computed in the classical way, i. e., the probabilities have no influence on that utility. As a consequence, a pattern may be of high-utility because some extremely improbable 2-tuples it covers have large utilities. In contrast, `multidupehack` enforces constraints on the ET-n-sets, and not on all-ones sub-tensors of a the tensor that would be obtained by turning every non-null membership degree into 1. As a consequence, the satisfaction of the constraints, in particular of the minimal-utility constraint, indirectly depends on the membership degrees, between 0 and 1.

All the cited algorithms enumerate itemsets, i. e., subsets of one dimension, by recursively adding one element to the last considered itemset, and refine an upper-bound of the maximal utility over all the itemsets (and their supporting sets) that may be recursively enumerated. If the upper-bound becomes less than the minimal-utility threshold, going on with the recursion is guaranteed to not lead to any high-utility itemset and the enumeration sub-tree is safely pruned. In contrast, this proposal relies on `multidupehack`, which employs different enumeration principles to list the ET-n-sets in an uncertain tensor, the itemsets in a 0/1 matrix being a special case. Indeed, `multidupehack` recursively adds to the previously considered pattern one element that can be taken in any of the n dimensions. Furthermore, `multidupehack` can prune the pattern space with any number of piecewise (anti-)monotone constraints. Section 5 will now show that the minimal-utility constraint is piecewise (anti-)monotone, even if some utilities are negative.

5. Pruning the pattern subspaces of low utility

Detailing `multidupehack`[1] is out of the scope of this article. However, a high-level presentation of that algorithm is useful to understand why and how it can prune the search of the patterns under any piecewise (anti-)monotone constraint, a property this section will soon define. Given an uncertain tensor, which maps every n-tuple $t \in \prod_{i=1}^{n} D_i$ to a value in $[0, 1]$, `multidupehack` recursively traverses the pattern space, i. e., the set of all sub-tensors, $\prod_{i=1}^{n} 2^{D_i}$. Every recursive call starts the exploration of a subspace that two patterns define: the lower bound $(L_1, \ldots, L_n)$ of the subspace and the upper bound $(U_1, \ldots, U_n)$ of the subspace. In other terms, a pattern $(X_1, \ldots, X_n)$ is in the subspace, that the two bounds define, if and only if $\forall i \in \{1, \ldots, n\}$, $L_i \subseteq X_i \subseteq U_i$. Initially, $(L_1, \ldots, L_n) = (\emptyset, \ldots, \emptyset)$ and $(U_1, \ldots, U_n) = (D_1, \ldots, D_n)$, i. e., `multidupehack` starts the exploration of the whole pattern space.

As illustrated in Figure 1, if $(L_1, \ldots, L_n) \neq (U_1, \ldots, U_n)$, an element $e \in \cup_{i=1}^{n} U_i \setminus L_i$ is selected and two recursive calls are made to start the exploration of two subspaces whose union contains all the ET-n-sets in the parent subspace: the patterns involving e (added to the respective dimension of the child lower bound) and the patterns that do not involve e (removed from the respective dimension of the child upper bound). The former subspace does not only have a "larger" (w.r.t. $\subseteq$ over all dimensions) lower bound than its parent. It usually has a "smaller" upper bound too: every element that the parent upper bound involves but that cannot extend the child lower bound without making it violate the definition of an ET-n-set is removed. In this way, any lower bound $(L_1, \ldots, L_n)$ is an ET-n-set and, if $(L_1, \ldots, L_n) = (U_1, \ldots, U_n)$, this ET-$n$-set is output.

Because it refines both a lower and an upper bound of the pattern space, `multidupehack` can prune the search of the patterns under any piecewise (anti-)monotone constraint. The piecewise (anti-)monotonicity relies on the notion of (anti-)monotonicity per argument. A constraint is (anti-)monotone w.r.t. its i^{th} argument if and only if, given subsets of the dimensions $D_1, \ldots, D_n$ satisfying it, the constraint (a) keeps on being satisfied it if the i^{th} subset is enlarged or (b) keeps on being satisfied if the i^{th} subset is shrunk. Formally: given $i \in \{1, \ldots, m\}$, a constraint C taking $m \in \mathbb{N}$

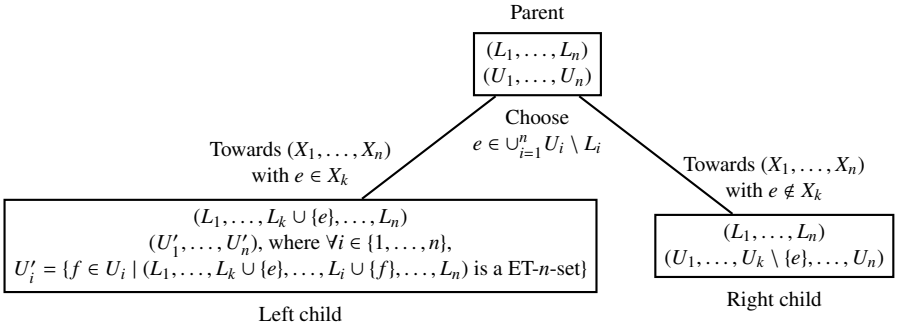


Fig. 1: `multidupehack`'s pattern space traversal.

subsets of the dimensions $D_1, \ldots, D_n$ is *(anti-)monotone w.r.t. its i^{th} argument*, a subset of one the dimensions D_k, if and only if $\begin{cases} \text{(monotonicity) } \forall X_i \subseteq D_k \text{ and } \forall Y_i \subseteq X_i, C(X_1, \ldots, X_{i-1}, Y_i, X_{i+1}, \ldots, X_m) \Rightarrow C(X_1, \ldots, X_m) \\ \text{or} \\ \text{(anti-monotonicity) } \forall X_i \subseteq D_k \text{ and } \forall Y_i \subseteq X_i, C(X_1, \ldots, X_m) \Rightarrow C(X_1, \ldots, X_{i-1}, Y_i, X_{i+1}, \ldots, X_m) \end{cases}$.

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

At first sight, $C_{\alpha\text{-min-utility}}$, defined in Section 2, does not look piecewise (anti-)monotone because there can be both positive and negative utilities. However, the addition being commutative, associative and 0 being its identity element, $C_{\alpha\text{-min-utility}}$ can be rewritten in this way:

$$(X_1, \ldots, X_n) \mapsto \sum_{\substack{t \in \prod_{i \in I} X_i \\ \text{such that } u(t)<0}} u(t) + \sum_{\substack{t \in \prod_{i \in I} X_i \\ \text{such that } u(t)>0}} u(t) \geq \alpha \ .$$

Attributing a separate argument to every occurrence of the variables $X_1, \ldots, X_n$ in the above expression results in the following constraint $C'_{\alpha\text{-min-utility}}$:

$$(X_1, \ldots, X_n, X'_1, \ldots, X'_n) \mapsto \sum_{\substack{t \in \prod_{i \in I} X_i \\ \text{such that } u(t)<0}} u(t) + \sum_{\substack{t \in \prod_{i \in I} X'_i \\ \text{such that } u(t)>0}} u(t) \geq \alpha \ .$$

$C'_{\alpha\text{-min-utility}}$ is anti-monotone w.r.t. to its first n arguments because shrinking any of these arguments removes negative utilities from the sum, which stays above α if it was above α. $C'_{\alpha\text{-min-utility}}$ is monotone w.r.t. its last n arguments because enlarging any of these arguments adds positive utilities to the sum, which stays above α if it was above α. $C'_{\alpha\text{-min-utility}}$ therefore is is (anti-)monotone w.r.t. each of its arguments and $C_{\alpha\text{-min-utility}}$ is piecewise (anti-)monotone.

That is why `multidupehack` can prune the search of the high-utility patterns: a new pattern subspace, defined by its lower bound $(L_1, \ldots, L_n)$ and its upper bound $(U_1, \ldots, U_n)$, is only explored if $(L_1, \ldots, L_n, U_1, \ldots, U_n)$ satisfies $C'_{\alpha\text{-min-utility}}$. Indeed, by (anti-)monotonicity of $C'_{\alpha\text{-min-utility}}$, if there exists a high-utility pattern $(X_1, \ldots, X_n)$ such that $\forall i \in \{1, \ldots, n\}$, $L_i \subseteq X_i \subseteq U_i$ (i. e., a pattern in the subspace), then $(L_1, \ldots, L_n, U_1, \ldots, U_n)$ necessarily satisfies $C'_{\alpha\text{-min-utility}}$. The pruning relies on the contraposition: after computing the bounds of a new pattern subspace, `multidupehack` tests whether $(L_1, \ldots, L_n, U_1, \ldots, U_n)$ violates $C'_{\alpha\text{-min-utility}}$ and, if so, leaves that space unexplored (i. e., the recursive traversal backtracks). Indeed, no pattern in it can possibly satisfy $C_{\alpha\text{-min-utility}}$.

Naively testing whether $\sum_{\substack{t \in \prod_{i \in I} L_i \\ \text{such that } u(t)<0}} u(t) + \sum_{\substack{t \in \prod_{i \in I} U_i \\ \text{such that } u(t)>0}} u(t) < \alpha$, i. e., whether $(L_1, \ldots, L_n, U_1, \ldots, U_n)$ violates $C'_{\alpha\text{-min-utility}}$, requires reading the utilities of all the $|\prod_{i \in I} U_i|$ $|I|$-tuples that the upper bound covers. Nevertheless, Figure 1 shows that the lower and upper bounds of a pattern subspace relate to the respective bounds of its parent subspace: $\forall i \in \{1, \ldots, n\}$, $L_i^{\text{child}} \supseteq L_i^{\text{parent}}$ and $U_i^{\text{child}} \subseteq U_i^{\text{parent}}$. Those properties make it easy to compute $\sum_{\substack{t \in \prod_{i \in I} L_i^{\text{child}} \\ \text{such that } u(t)<0}} u(t) + \sum_{\substack{t \in \prod_{i \in I} U_i^{\text{child}} \\ \text{such that } u(t)>0}} u(t)$ as an update of $\sum_{\substack{t \in \prod_{i \in I} L_i^{\text{parent}} \\ \text{such that } u(t)<0}} u(t) + \sum_{\substack{t \in \prod_{i \in I} U_i^{\text{parent}} \\ \text{such that } u(t)>0}} u(t)$, which was computed to

test whether parent pattern subspace could contain a high-utility pattern:

$$\sum_{\substack{t\in\prod_{i\in I} L_i^{\text{parent}} \\ \text{such that } u(t)<0}} u(t) + \sum_{\substack{t\in\prod_{i\in I} U_i^{\text{parent}} \\ \text{such that } u(t)>0}} u(t) + \sum_{\substack{t\in\left(\prod_{i\in I} L_i^{\text{child}}\right)\setminus\left(\prod_{i\in I} L_i^{\text{parent}}\right) \\ \text{such that } u(t)<0}} u(t) + \sum_{\substack{t\in\left(\prod_{i\in I} U_i^{\text{parent}}\right)\setminus\left(\prod_{i\in I} U_i^{\text{child}}\right) \\ \text{such that } u(t)>0}} u(t) \ .$$

Thanks to that incremental computation, only $\left|\left(\prod_{i\in I} L_i^{\text{child}}\right) \setminus \left(\prod_{i\in I} L_i^{\text{parent}}\right)\right| + \left|\left(\prod_{i\in I} U_i^{\text{parent}}\right) \setminus \left(\prod_{i\in I} U_i^{\text{child}}\right)\right|$ $|I|$-tuples are read after every recursive call of `multidupehack`, hence a smaller computational cost than the naive implementation.

6. Experimental validation

`multidupehack`, augmented with the minimal-utility constraint, is free software written in C++ and available under the terms of the GNU GPLv3 at `http://dcc.ufmg.br/~lcerf/en/prototypes.html#multidupehack`. To compile it, `g++` 5.4 was used at the O3 optimization level. Java 8 runs the SPMF[18] implementations of the competing algorithms. All the experiments are performed on a GNU/Linux™ system running on top of a 3.5 GHz core (all implementations are monothreaded). Missing points on a curve relate to executions that require more than 10 GB of RAM or more than two hours of computation.

6.1. High-utility itemsets in 0/1 matrices: comparison with the state of art

Mining high-utility itemsets, with no tolerance to noise and only positive utilities is a well-studied problem. FHM[4], EFIM[5], ULB-Miner[6] and mHUIMiner[7] are the fastest existing algorithms to solve it. `multidupehack` is compared to them in that specific context. Four 0/1 matrices are used: `chess` of size $3{,}196 \times 75$, `connect` of size $67{,}557 \times 129$, `foodmart` of size $3{,}196 \times 75$, and `mushroom` of size $8{,}124 \times 119$. They were all downloaded from SPMF's website, which provides as well positive utilities for them, sampled from a uniform distribution in $[0, 10]$, except `foodmart`'s utilities, which are not synthetic.

Figure 2 reports run times in function of the minimal utility threshold, α, varying in intervals commonly adopted in the literature, which uses the same datasets. Despite `multidupehack`'s far broader scope, its performances are not shameful. EFIM is the fastest contender on all the tested datasets but `foodmart`, where `multidupehack` does not perform well either. For each dataset, Figure 3 shows how much memory every competitor requires, in average over the tested settings. `multidupehack` has, by a large margin, the smallest memory requirements to mine `chess` and `mushroom`. `connect` is the most memory-demanding dataset. `multidupehack` mines it consuming at most 680 MB, whereas FHM and mHUIMiner require more than 1.6 GB. For each dataset, Figure 4 reports, in average over the tested settings, the numbers of high-utility itemsets that are closed in both dimensions (`multidupehack (closed)`, in Figure 2, lists them) or only closed in the supporting set (all the other algorithms, in Figure 2, list them). On `mushroom`, where less than one percent of the high-utility itemsets are closed, `multidupehack (closed)` is more than 200 times faster than `multidupehack`. That makes it the second best performer, whereas `multidupehack` is the worse. As explained in Section 4, an analyst would be probably only interested in the *closed* patterns because, every utility being here positive, their sub-patterns necessarily have smaller utilities.

CHUI-Miner[13] and EFIM-Closed[14] specifically mine *closed* high-utility itemsets, still with no tolerance to noise and only positive utilities, in 0/1 matrices. Figure 5 shows their run times along those of `multidupehack` and `multidupehack (closed)`. Although EFIM-Closed is usually the fastest, it is much slower than EFIM (in Figure 2) on `chess` and `connect`. `multidupehack (closed)` is the best performer on `connect`. Figure 5 includes as well a curve named `multidupehack (closed and size constraints)`. It relates to experiments where `multidupehack` lists the closed high-utility patterns that involve at least $\gamma \in \mathbb{N}$ elements of each of the two dimensions. Who wants to only interpret large pattern desires such a constraint that none of the competitors handles. In `foodmart`, for instance, it filters out many high-utility patterns that involve one single row. γ is here set to a value that depends on the dataset but that at least one high-utility pattern satisfies in the most constrained setting: 17 for `connect` and `mushroom`, 15 for chess, and 3 for `foodmart`. The additional constraints greatly decrease the run times: `multidupehack (closed and size constraints)` is almost always the fastest competitor in Figure 5.

HUINIV-Mine[9], FHN[10] and, of course, `multidupehack`, can mine the high-utility itemsets (with no tolerance to noise) in 0/1 matrices whose cells are associated with positive *and* negative utilities. SPMF's website hosts such

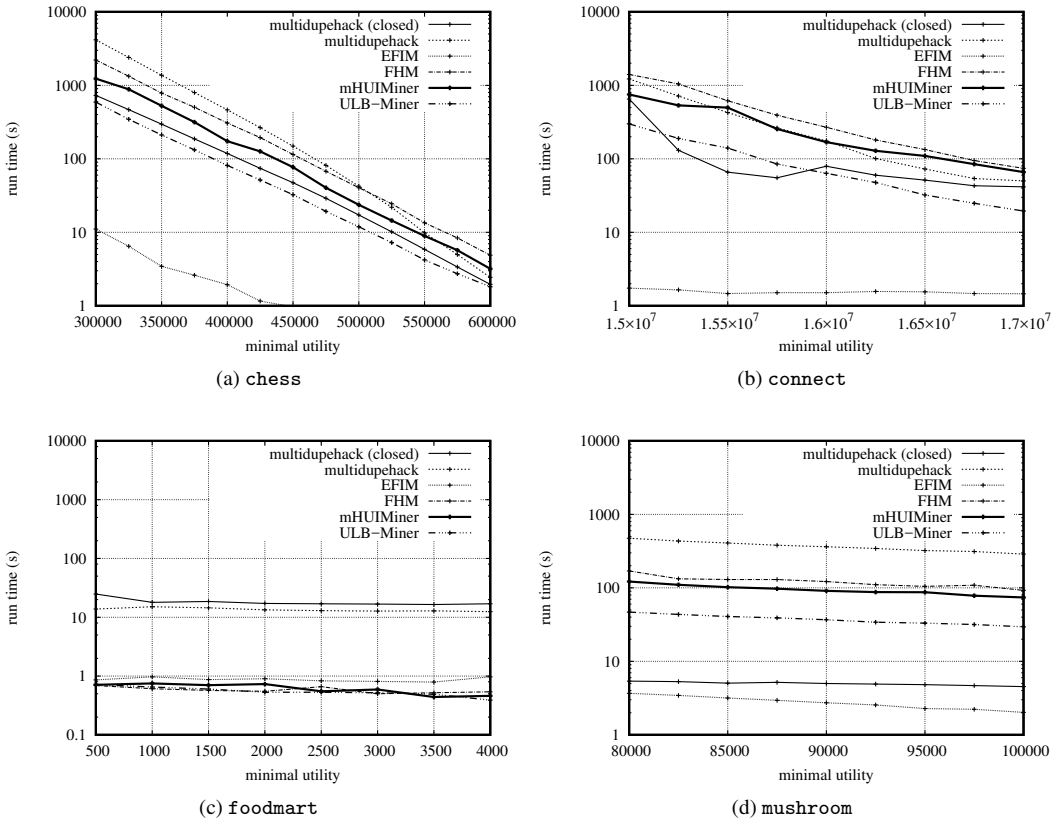


Fig. 2: Times to list the high-utility itemsets, in presence of no negative utility.

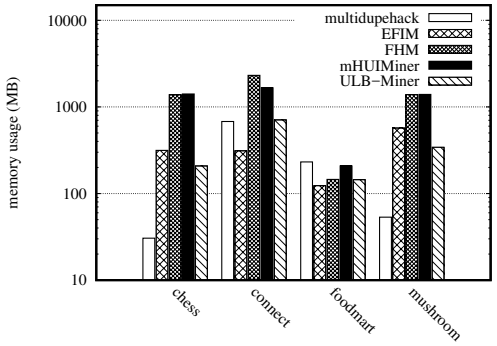


Fig. 3: Maximal memory consumption over the tested settings.

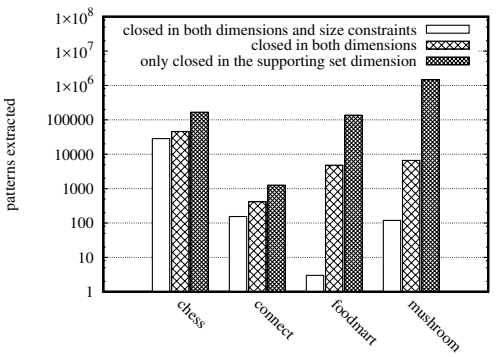


Fig. 4: Average numbers of patterns over the tested settings.

datasets but does not provide a version of `connect` with negative utilities. `foodmart`'s real-world utilities are all positive too. That is why `accidents`, of size 340,183×468, and `pumsb`, of size 49,046×2,113, now replace `connect` and `foodmart`. Although `multidupehack` solves a far more general problem, with possibly more dimensions of analysis and a tolerance to noise, it compares well against HUINIV-Mine and FHN, which are specifically designed to list high-utility itemsets in presence of positive *and* negative utilities. Figure 6 shows that `multidupehack`, when it forces the patterns to be closed in both dimensions, is the fastest to process `chess` and `mushroom`. On `mushroom`, it is

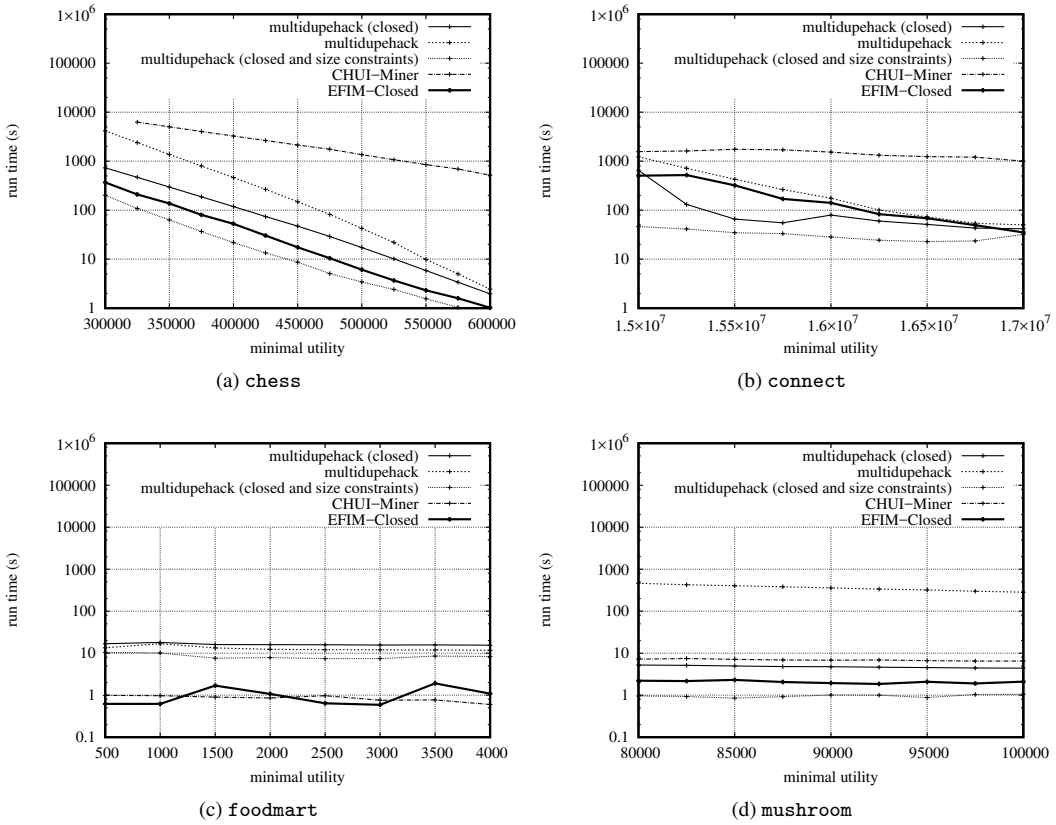


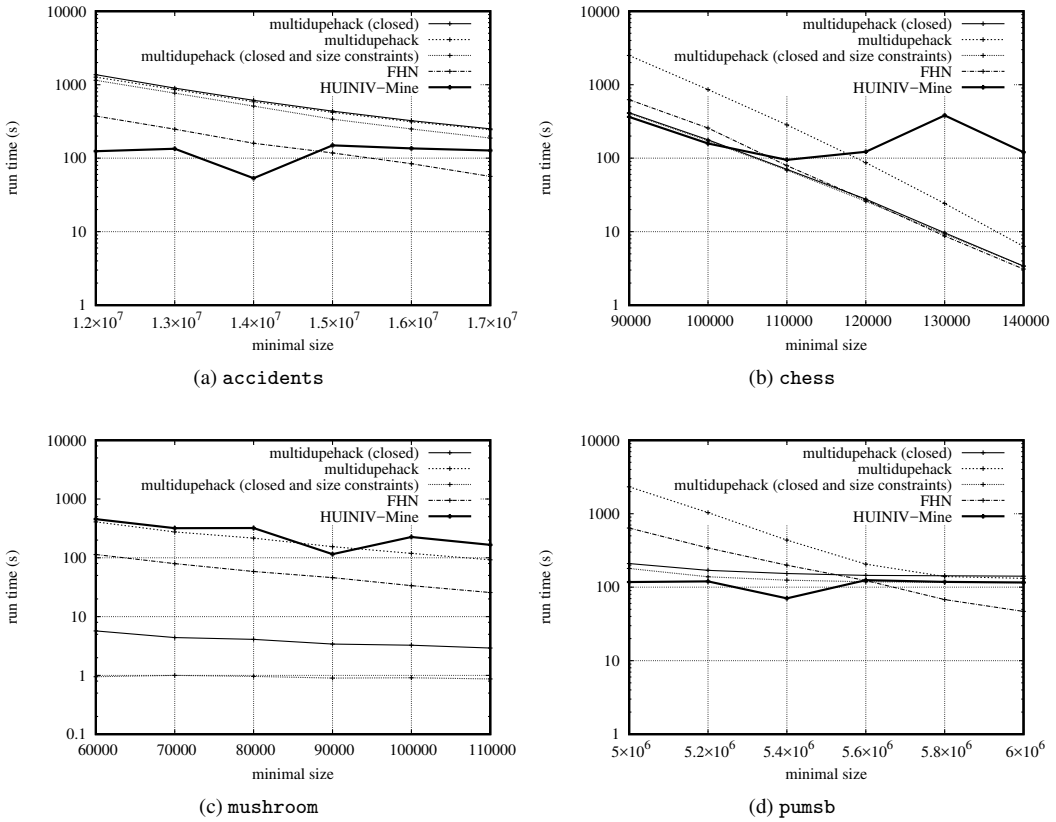
Fig. 5: Times to list the *closed* high-utility itemsets, in presence of no negative utility.

up to twenty times faster than FHN, the second best performer. As before, the curve `multidupehack (closed and size constraints)` is even lower. It reports the results of experiments where the the closed high-utility patterns are additionally forced to involve at least $\gamma \in \mathbb{N}$ elements of each of the two dimensions. This time, γ is set to 7 for `accidents` and `chess`, 16 for `mushroom`, and 3 for `pumsb`.

6.2. Constrained closed high-utility patterns in a real-world uncertain tensor

A 3-way uncertain tensor is built from the connection and disconnection times of spectators who watched video streams on `twitch.tv`, a Web TV specialized on video games. The tensor is large: 1,198,282 spectators (first dimension) watched at least one of the 94 collected channels (second dimension) during the 19 weeks (third dimension) of the data collect, from October 7$^{\text{th}}$ 2013 to February 16$^{\text{th}}$ 2014. The channels were selected for focusing on the StarCraft II video game and for having reached at least 1,000 simultaneous visualizations during the week that preceded the data collect. Every membership degree in the uncertain tensor aims to quantify to what extent a spectator enjoyed a channel during a week. The total time (in seconds) she spent on that channel during the week is used as a proxy. A logistic function, centered on 3,600 seconds (spending one hour was enjoying the channel moderately: $T_t = 0.5$) and with a 0.0015 growth rate, turns such a time into a membership degree.

Those same times are used as utilities (with $I = \{1, 2, 3\}$). To list a reasonable number of patterns in a large dataset, such as the `twitch.tv` tensor, these patterns must be strongly constrained. In addition to having a high utility, the patterns are here required to involve at least three spectators, three channels and three weeks. The minimal utility threshold, α, is set to 27,600,000 seconds ($\approx$ 319 days). Such a threshold can be, and was, iteratively fixed: decreasing values for α are tested until `multidupehack`'s output is not empty. High minimal utility thresholds entailing fast executions, that process is not prohibitively time consuming. Of course, the patterns must be ET-3-sets



(a) `accidents` (b) `chess`

(c) `mushroom` (d) `pumsb`

Fig. 6: Times to list the high-utility itemsets, in presence of positive and negative utilities.

too. ϵ_1, ϵ_2 and ϵ_3 are all set to 1. Finally, only *closed* ET-3-sets are listed. It takes `multidupehack` 51 minutes and 42 seconds to list the 96,701 patterns satisfying all the requirements. Additionally forcing any two subsequent weeks, involved in a pattern, to be separated by at most one (missing) week (Cerf and Meira formally define this constraint[1]), `multidupehack`'s run time becomes 31 minutes and 58 seconds, hence a division by 1.9. Surprisingly, the same patterns are listed: the weeks involved in any closed ET-3-set satisfying the remaining constraints are always almost-contiguous. The ET-3-set of highest utility involves 83 spectators who each spent, in average, 3.9 days watching the channels `mlgsc2`, `wcs_america` and `wcs_europe2` during the first three weeks of the collect. That pattern makes sense: those three channels broadcast the American and European games of the 2013 *StarCraft II World Championship Series*, the main StarCraft II tournament, which ended at the end of the last week involved in the pattern. A fourth channel, `wcs_gsl`, broadcast the Korean games but they happened before the collect.

Instead of attaching utilities to the 3-tuples, the utility function can take a pair (`channel`, `week`) at input (i. e., $I = \{2, 3\}$) and return the maximal number of spectators who were simultaneously watching the `channel` over the `week`. As a consequence, the minimal-utility constraint forces the channels in a returned pattern to have been "popular" during the weeks in this same pattern. Using the same configuration as above (including the almost-contiguity constraint), except for the minimal utility, which is set to 75,000 spectators, `multidupehack` now takes 6 minutes and 14 seconds to list 102 closed ET-3-sets. Again: they make sense. For instance, among all the returned patterns, those that involve the last three weeks of the collect, from January 27$^{\text{th}}$ 2014 to February 16$^{\text{th}}$ 2014, always involve `egstephano` too. That channel, which broadcast games of the eponymous player, does not occur in any pattern involving the previous weeks. Stephano indeed announced, months before the beginning of the collect, its retirement from StarCraft II competitions but he unexpectedly took part in the qualifications for the *2014 WCS Season 2 Europe* and defeated the three players he played against, at the end of January 2014.

7. Conclusion

Unlike the existing literature, this article has not proposed an algorithm that is specifically designed to mine high-utility itemsets. Here, "having a utility exceeding a threshold" has been seen as a constraint, which has been proven piecewise (anti-)monotone, even in presence of both positive and negative utilities. Thanks to that property, a generic algorithm, `multidupehack`, can prune the search of the high-utility patterns defined in a broad context: the uncertain tensor. That allows the analyst to take into consideration any number n of dimensions of analysis and to quantify to what extent every n-tuple satisfies the Boolean predicate that the tensor encodes. Moreover, she can now force the high-utility patterns to be closed in any subset of the dimensions and to satisfy additional piecewise (anti-)monotone constraints. Choosing a conjunction of such constraints is analog to designing a database query: every returned pattern/record must satisfy it. Going on with that analogy, this article has brought SQL's `WHERE SUM` $\geq \alpha$ to pattern mining. Although the proposal addresses a general problem, it has been shown competitive when it comes to mining high-utility itemsets in 0/1 matrices, i. e., the specific problem tackled by dozens of existing algorithms.

Acknowledgements

The work has been partially funded by the FAPEMIG under Grants № APQ-01400-14 (FAPEMIG-PRONEX-MASWeb) and APQ-04224-16 (Multilateral Cooperation FAPEMIG-CNRS). It is also supported by the EUBra-BIGSEA project (690116).

References

1. Cerf, L., Meira, W.. Complete discovery of high-quality patterns in large numerical tensors. In: *ICDE*. IEEE Computer Society; 2014, p. 448–459.
2. Ng, R.T., Lakshmanan, L.V., Han, J., Pang, A.. Exploratory mining and pruning optimizations of constrained associations rules. In: *ACM Sigmod Record*; vol. 27. ACM; 1998, p. 13–24.
3. Yao, H., Hamilton, H.J., Butz, C.J.. A foundational approach to mining itemset utilities from databases. In: *Proc. of the 2004 SIAM Int. Conf. on Data Mining*. SIAM; 2004, p. 482–486.
4. Fournier-Viger, P., Wu, C.W., Zida, S., Tseng, V.S.. FHM: Faster high-utility itemset mining using estimated utility co-occurrence pruning. In: *Int. symposium on methodologies for intelligent systems*. Springer; 2014, p. 83–92.
5. Zida, S., Fournier-Viger, P., Lin, J.C.W., Wu, C.W., Tseng, V.S.. EFIM: a highly efficient algorithm for high-utility itemset mining. In: *Mexican Int. Conf. on Artificial Intelligence*. Springer; 2015, p. 530–546.
6. Duong, Q.H., Fournier-Viger, P., Ramampiaro, H., Nørvåg, K., Dam, T.L.. Efficient high utility itemset mining using buffered utility-lists. *Applied Intelligence* 2017;.
7. Peng, A.Y., Koh, Y.S., Riddle, P.. mHUIMiner: A fast high utility itemset mining algorithm for sparse datasets. In: *Advances in Knowl. Discovery and Data Mining*; Lecture Notes in Computer Science. Springer, Cham; 2017, p. 196–207.
8. Fournier-Viger, P., Lin, J.C.W., Duong, Q.H., Dam, T.L.. FHM+ : Faster high-utility itemset mining using length upper-bound reduction. In: Fujita, H., Ali, M., Selamat, A., Sasaki, J., Kurematsu, M., editors. *Trends in Applied Knowledge-Based Systems and Data Science*; vol. 9799. Cham: Springer; 2016, p. 115–127.
9. Chu, C.J., Tseng, V.S., Liang, T.. An efficient algorithm for mining high utility itemsets with negative item values in large databases. *Applied Mathematics and Computation* 2009;**215**(2):767–778.
10. Fournier-Viger, P.. FHN: efficient mining of high-utility itemsets with negative unit profits. In: *Int. Conf. on Advanced Data Mining and Applications*. Springer; 2014, p. 16–29.
11. Krishnamoorthy, S.. Efficiently mining high utility itemsets with negative unit profits. *Knowledge-Based Systems* 2017;.
12. Tseng, V.S., Wu, C.W., Fournier-Viger, P., Philip, S.Y.. Efficient algorithms for mining the concise and lossless representation of high utility itemsets. *IEEE Transactions on Knowledge and Data Engineering* 2015;**27**(3):726–739.
13. Wu, C.W., Fournier-Viger, P., Gu, J.Y., Tseng, V.S.. Mining closed+ high utility itemsets without candidate generation. In: *Technologies and Applications of Artificial Intelligence (TAAI), 2015 Conference on*. IEEE; 2015, p. 187–194.
14. Fournier-Viger, P., Zida, S., Lin, J.C.W., Wu, C.W., Tseng, V.S.. EFIM-Closed: fast and memory efficient discovery of closed high-utility itemsets. In: *Machine Learning and Data Mining in Pattern Recognition*. Springer; 2016, p. 199–213.
15. Lin, J.C.W., Gan, W., Fournier-Viger, P., Hong, T.P., Tseng, V.S.. Efficient algorithms for mining high-utility itemsets in uncertain databases. *Knowledge-Based Systems* 2016;**96**:171–187.
16. Lin, J.C.W., Gan, W., Fournier-Viger, P., Hong, T.P., Tseng, V.S.. Efficiently mining uncertain high-utility itemsets. *Soft Computing* 2017; **21**(11):2801–2820.
17. Gan, W., Lin, J.C.W., Fournier-Viger, P., Chao, H.C., Tseng, V.S.. Mining high-utility itemsets with both positive and negative unit profits from uncertain databases. In: Kim, J., Shim, K., Cao, L., Lee, J.G., Lin, X., Moon, Y.S., editors. *Advances in Knowl. Discovery and Data Mining*. Cham: Springer; 2017, p. 434–446.
18. Fournier-Viger, P., Gomariz, A., Gueniche, T., Soltani, A., Wu, C.W., Tseng, V.S.. SPMF: a Java open-source pattern mining library. *The Journal of Machine Learning Research* 2014;**15**(1):3389–3393.