

Flexible constrained sampling with guarantees for pattern mining

Vladimir Dzyuba¹  · Matthijs van Leeuwen² ·
Luc De Raedt¹

Received: 28 August 2016 / Accepted: 3 March 2017 / Published online: 24 March 2017
© The Author(s) 2017

Abstract Pattern sampling has been proposed as a potential solution to the infamous pattern explosion. Instead of enumerating all patterns that satisfy the constraints, individual patterns are sampled proportional to a given quality measure. Several sampling algorithms have been proposed, but each of them has its limitations when it comes to (1) flexibility in terms of quality measures and constraints that can be used, and/or (2) guarantees with respect to sampling accuracy. We therefore present FLEXICS, the first flexible pattern sampler that supports a broad class of quality measures and constraints, while providing strong guarantees regarding sampling accuracy. To achieve this, we leverage the perspective on pattern mining as a constraint satisfaction problem and build upon the latest advances in sampling solutions in SAT as well as existing pattern mining algorithms. Furthermore, the proposed algorithm is applicable to a variety of pattern languages, which allows us to introduce and tackle the novel task of sampling sets of patterns. We introduce and empirically evaluate two variants of FLEXICS: (1) a generic variant that addresses the well-known itemset sampling task and the novel pattern set sampling task as well as a wide range of expressive constraints within these tasks, and (2) a specialized variant that exploits existing frequent itemset techniques

Responsible editor: Kurt Driessens, Dragi Koccev, Marko Robnik-Šikonja and Myra Spiliopoulou.

✉ Vladimir Dzyuba
vladimir.dzyuba@cs.kuleuven.be

Matthijs van Leeuwen
m.van.leeuwen@liacs.leidenuniv.nl

Luc De Raedt
luc.deraedt@cs.kuleuven.be

¹ Department of Computer Science, KU Leuven, Leuven, Belgium

² LIACS, Leiden University, Leiden, The Netherlands

to achieve substantial speed-ups. Experiments show that FLEXICS is both accurate and efficient, making it a useful tool for pattern-based data exploration.

Keywords Pattern sampling · Itemset mining · Pattern set mining · Tiling · Hashing-based sampling

1 Introduction

Pattern mining (Agrawal et al. 1996) is an important and well-studied task in data mining. Informally, a pattern is a statement in a formal language that concisely describes a subset of a given dataset. Pattern mining techniques aim at providing comprehensible descriptions of coherent regions in the data. Many variations of pattern mining have been proposed in the literature, together with even more algorithms to efficiently mine the corresponding patterns. Best known is frequent pattern mining (Agrawal and Han 2014), which includes frequent itemset mining and its extensions.

Traditional pattern mining methods enumerate all frequent patterns, though it is well-known that this usually results in humongous amounts of patterns (the infamous *pattern explosion*). To make pattern mining more useful for exploratory purposes, different solutions to this problem have been proposed. Each of these solutions has its own advantages and disadvantages. *Condensed representations* (Calders et al. 2006) can often be efficiently mined, but generally still result in large numbers of patterns. *Top-k mining* (Zimmermann and Nijssen 2014) is efficient but results in strongly related, redundant patterns showing a lack of diversity. *Constrained mining* (Nijssen and Zimmermann 2014) may result in too few or too many patterns, depending on the user-chosen constraints. *Pattern set mining* (Bringmann et al. 2010) takes into account the relationships between the patterns, which can result in small solution sets, but is computationally intensive.

In this paper, we study *pattern sampling*, another approach that has been proposed recently: instead of enumerating all patterns, patterns are sampled one by one, according to a probability distribution that is proportional to a given quality measure. The promised benefits include: (1) flexibility in that potentially a broad range of quality measures and constraints can be used; (2) ‘anytime’ data exploration, where a growing representative set of patterns can be generated and inspected at any time; (3) diversity in that the generated sets of patterns are independently sampled from different regions in the solution space. To be reliable, pattern samplers should provide theoretical guarantees regarding the sampling accuracy, i.e., the difference between the empirical probability of sampling a pattern and the (generally unknown) target probability determined by its quality. These properties are essential for pattern mining applications ranging from showing patterns directly to the user, where flexibility and the anytime property enable experimenting with and fine-tuning mining task formulations, to candidate generation for building pattern-based models, for which the approximation guarantees can be derived from those of the sampler.

While a number of pattern sampling approaches have been developed over the past years, they are either inflexible (as they only support a limited number of quality measures and constraints), or do not provide theoretical guarantees concerning the

sampling accuracy. At the algorithmic level, they follow standard sampling approaches such as Markov Chain Monte Carlo random walks over the pattern lattice (Boley and Grosskreutz 2009; Hasan and Zaki 2009; Boley et al. 2010), or a special purpose sampling procedure tailored for a restricted set of itemset mining tasks (Boley et al. 2011, 2012). Although MCMC approaches are in principle applicable to a broad range of tasks, they often converge only slowly to the desired target distribution and require the selection of the “right” proposal distributions.

To the best of our knowledge, none of the existing approaches to pattern sampling takes advantage of the latest developments in sampling technology from the SAT-solving community, where a number of powerful samplers based on random hash functions and XOR-sampling have been developed (Gomes et al. 2007b; Chakraborty et al. 2013; Ermon et al. 2013a; Meel et al. 2016). WEIGHTGEN (Chakraborty et al. 2014), one of the recent approaches, possesses the benefits mentioned above: it is an anytime algorithm, it is flexible as it works with any distribution, it generates diverse solutions, and provides strong performance guarantees under reasonable assumptions.

In this paper, we show that the latest developments in sampling solutions in SAT are also relevant to pattern sampling and essentially offer the same advantages. Our results build upon the view of pattern mining as constraint satisfaction, which is now commonly accepted in the data mining community (Guns et al. 2011).

1.1 Approach and contributions

More specifically, we introduce FLEXICS: a flexible pattern sampler that samples from distributions induced by a variety of pattern quality measures and allows for a broad range of constraints while still providing strong theoretical guarantees. Notably, FLEXICS is, in principle, agnostic of the quality measure, as the sampler treats it as a black box. (However, its properties affect the efficiency of the algorithm.) The other building block is a *constraint oracle* that enumerates all patterns that satisfy the constraints, i.e., a mining algorithm. The proposed approach allows converting an existing pattern mining algorithm into a sampler with guarantees. Thus, its flexibility is not limited by the choice of constraints and quality measures, but even allows tackling richer pattern languages, which we demonstrate by tackling the novel task of *sampling sets of patterns*. Table 1 compares the proposed approach to alternative samplers; see Sect. 3 for a more detailed discussion.

The main technical contribution of this paper consists of two variants of the FLEXICS sampler, which are based on different constraint oracles. First, we introduce a generic variant, dubbed GFLEXICS, that supports a wide range of pattern constraints, such as syntactic or redundancy-eliminating constraints. GFLEXICS uses CP4IM (Guns et al. 2011), a declarative constraint programming-based mining system, as its oracle. Any constraint supported by CP4IM can be used without interfering with the umbrella procedure that performs the actual sampling task. Unlike the original version of WEIGHTGEN that is geared towards SAT, GFLEXICS can handle cardinality constraints that are ubiquitous in pattern mining. Furthermore, we identify (based on previous research) the properties of the constraint satisfaction-based formalization of pattern mining that further improve the efficiency of the sampling procedure without affecting its guarantees

Table 1 Our method is the first pattern sampler that combines flexibility with respect to the choice of constraints and sampling distributions with strong theoretical guarantees

Sampler	Arbitrary constraints	Arbitrary distributions	Strong guarantees	Efficiency	Pattern set sampling
ACFI Boley and Grosskreutz (2009)	Minimal frequency	–	–	✓	–
LRW Hasan and Zaki (2009)	✓	✓	–	Implementation-specific	–
FCA Boley et al. (2010)	Anti-/monotonic	✓	–	✓	–
TS (two-step) Boley et al. (2011, 2012)	–	–	✓	✓	–
FLEXICS This paper	GFLEXICS	✓	✓	EFLEXICS	✓

and thus make it applicable to practical problems. We use GFLEXICS to tackle a wide range of well-known itemset sampling tasks as well as the novel pattern set sampling task. Second, as it is well-known that generic solvers impose an overhead on runtime, we introduce a variant specialized towards frequent itemsets, dubbed EFLEXICS, which has an extended version of ECLAT (Zaki et al. 1997) at its core as oracle.

Experiments show that FLEXICS' sampling accuracy is impressively high: in a variety of settings supported by the sampler, empirical frequencies are within a small factor of the target distribution induced by various quality measures. Furthermore, practical accuracy is substantially higher than theory guarantees. EFLEXICS is shown to be faster than its generic cousin, demonstrating that developing specialized solvers for specific tasks is beneficial when runtime is an issue. Finally, the flexibility of the sampler allows us to use the same approach to successfully tackle the novel problem of sampling pattern sets. This demonstrates that FLEXICS is a useful tool for pattern-based data exploration.

This paper is organized as follows. We formally define the problem of pattern sampling in Sect. 2. After reviewing related research in Sect. 3, we present the two key ingredients of the proposed approach in Sect. 4: (1) the perspective on pattern mining as a constraint satisfaction problem and (2) hashing-based sampling with WEIGHTGEN. In Sect. 5, we present FLEXICS, a flexible pattern sampler with guarantees. In particular, we outline the modifications required to adapt WEIGHTGEN to pattern sampling and describe the procedure to convert two existing mining algorithms into oracles suitable for use with WEIGHTGEN, which yields two variants of FLEXICS. In Sect. 6, we introduce the pattern set sampling task and describe how it can be tackled with FLEXICS. We also outline sampling non-overlapping tilings, an example of pat-

tern set sampling that is studied in the experiments. The experimental evaluation in Sect. 7 investigates the accuracy, scalability, and flexibility of the proposed sampler. We discuss its potential applications, advantages, and limitations in Sect. 8. Finally, we present our conclusions in Sect. 9.

2 Problem definition

Here we present a high-level definition of the task that we consider in this paper; for concrete instances and examples, see Sects. 4 and 6. The pattern sampling problem is formally defined as follows: given a dataset \mathcal{D} , a pattern language \mathcal{L} , a set of constraints \mathcal{C} , and a quality measure $\varphi: \mathcal{L} \rightarrow \mathbb{R}^+$, generate random patterns that satisfy constraints in \mathcal{C} with probability proportional to their qualities:

$$P_{\varphi}(p) = \begin{cases} \varphi(p) / Z_{\varphi} & \text{if } p \in \mathcal{L} \text{ satisfies } \mathcal{C} \\ 0 & \text{otherwise} \end{cases}$$

where Z_{φ} is an (often unknown) normalization constant.

A quality measure quantifies the domain-specific interestingness of a pattern. The choice of a quality measure and constraints allows a user to express her analysis requirements. The sampling procedure meets these requirements by satisfying the constraints and generating high-quality patterns more frequently. Thus, sampled patterns are a representative subset of all interesting regularities in the dataset.

Pattern set mining is an extension of pattern mining, which considers sets of patterns rather than individual patterns. Despite its popularity, we are not aware of the existence of pattern set samplers. The task of pattern set sampling can easily be formalized as an extension of pattern sampling, where we sample sets of patterns $s \subset \mathcal{L}$, and the constraints \mathcal{C} as well as the quality measure φ are specified over sets of patterns (from $2^{\mathcal{L}}$) rather than individual patterns (from \mathcal{L}).

3 Related work

We here focus on two classes of related work, i.e., (1) pattern mining as constraint satisfaction and (2) pattern sampling.

3.1 Constrained pattern mining

The study of constraints has been a prominent subfield of pattern mining. A wide range of constraint classes were investigated, including anti-monotonic constraints (Agrawal et al. 1996), convertible constraints (Pei and Han 2000), and others. Another development of these ideas led to the introduction of global constraints that concern multiple patterns and to the emergence of *pattern set mining* (Knobbe and Ho 2006; De Raedt and Zimmermann 2007). Furthermore, generic mining systems that could

freely combine various constraints were proposed (Bucilă et al. 2003; Bonchi et al. 2009).

These insights allowed to draw a connection between pattern mining and constraint satisfaction in AI, e.g., SAT or constraint programming (CP). As a result, declarative mining systems, which use generic constraint solvers to mine patterns according to a declarative specification of the mining task, were proposed. For example, CP was used to develop first declarative systems for itemset mining (Guns et al. 2011) and pattern set mining (Khiari et al. 2010; Guns et al. 2013). Recently, declarative approaches have been extended to support sequence mining (Kemmar et al. 2014) and graph mining (Paramonov et al. 2015).

Constraint-based systems allow a user to specify a wide range of pattern constraints and thus provide tools to alleviate the pattern explosion. However, the underlying solvers use systematic search, which affects the order of pattern generation and thus prevents them from being used in a truly anytime manner due to low diversity of consecutive solutions. Similarly, pattern set miners that directly aim at obtaining diverse result sets typically incur prohibitive computational costs as the size of the pattern space grows.

3.2 Pattern sampling

In this paper we focus on the approaches that directly aim at generating random pattern collections rather than the methods whose goal is to estimate dataset or pattern language statistics; cf. Shervashidze et al. (2009).

Table 1 compares our method with the approaches described in Sect. 1, namely MCMC and *two-step* samplers (Boley et al. 2011, 2012). We further break down MCMC samplers into three groups: ACFI, the very first uniform sampler developed for approximate counting of frequent itemsets (Boley and Grosskreutz 2009); LRW, a generic approach based on random walks over pattern lattice (Hasan and Zaki 2009); and FCA, a sampler, which uses Markov chains based on insights from formal concept analysis (Boley et al. 2010).

Although MCMC samplers provide theoretical guarantees, in practice, their convergence is often slow and hard to diagnose. Solutions such as long burn-in or heuristic adaptations either increase the runtime or weaken the guarantees. Furthermore, ACFI is tailored for a single task; FCA only supports anti-/monotone constraints; and LRW checks constraints locally, while building the neighborhood of a state, which might require advanced reasoning and extensive caching. Two-step samplers, while provably accurate and efficient, only support a limited number of weight functions and do not support constraints.

4 Preliminaries

We first outline itemset mining, a prototypical pattern mining task, and formalize it as a CSP and then describe WEIGHTGEN, a hashing-based sampling algorithm.

Table 2 Constraint programming formulations of common itemset mining constraints

Constraint	Parameters	CP formulation
<i>coverage</i>		$\forall t \in \mathcal{T} \quad T_t = 1 \Leftrightarrow \sum_{i \in \mathcal{I}} I_i (1 - \mathcal{D}_{ti}) = 0$
<i>minfreq</i> (θ)	$\theta \in (0, 1]$	$\forall i \in \mathcal{I} \quad I_i = 1 \Rightarrow \sum_{t \in \mathcal{T}} T_t \mathcal{D}_{ti} \geq \theta \times \mathcal{D} $
<i>closed</i>		$\forall i \in \mathcal{I} \quad I_i = 1 \Leftrightarrow \sum_{t \in \mathcal{T}} T_t (1 - \mathcal{D}_{ti}) = 0$
<i>minlen</i> (λ)	$\lambda \in [1, M]$	$\forall t \in \mathcal{T} \quad T_t = 1 \Rightarrow \sum_{i \in \mathcal{I}} I_i \mathcal{D}_{ti} \geq \lambda$

$I_i = 1$ implies that item i is included in the current (partial) solution, whereas $T_t = 1$ implies that it covers transaction t

4.1 Itemset mining

Itemset mining is an instance of pattern mining specialized for binary data. Let $\mathcal{I} = \{1 \dots M\}$ denote a set of items. A dataset \mathcal{D} is a bag of transactions over \mathcal{I} , where each transaction t is a subset of \mathcal{I} , i.e., $t \subseteq \mathcal{I}$; $\mathcal{T} = \{1 \dots N\}$ is a set of transaction indices. The pattern language \mathcal{L} also consists of sets of items, i.e., $\mathcal{L} = 2^{\mathcal{I}}$. An itemset p occurs in a transaction t , iff $p \subseteq t$. The frequency of p is the number of transactions in which it occurs: $\text{freq}(p) = |\{t \in \mathcal{D} \mid p \subseteq t\}|$. In labeled datasets, a transaction has a label from $\{-, +\}$; $\text{freq}^{-,+}$ are defined accordingly.

We first give a brief overview of the general approach to solving CSPs and then present a formalization of itemset mining as a CSP, following that of CP4IM (Guns et al. 2011). Formally, a CSP is comprised of *variables* along with their *domains* and *constraints* over these variables. The goal is to find a solution, i.e., an assignment of values to all variables that satisfies all constraints. Every constraint is implemented by a *propagator*, i.e., an algorithm that takes domains as input and removes values that do not satisfy the constraint. Propagators are activated when variable domains change, e.g., by the search mechanism or other propagators. A CSP solver is typically based on depth-first search. After a variable is assigned a value, propagators are run until domains cannot be reduced any further. At this point, three cases are possible: (1) a variable has an empty domain, i.e., the current search branch has failed and backtracking is necessary, (2) there are unassigned variables, i.e., further branching is necessary, or (3) all variables are assigned a value, i.e., a solution is found.

Let I_i denote a variable corresponding to each item; T_t a variable corresponding to each transaction; and \mathcal{D}_{ti} a constant that is equal to 1, if item i occurs in transaction t , and 0 otherwise. Variables I_i and T_t are *binary*, i.e., their domain is $\{0, 1\}$. Each CSP solution corresponds to a single itemset. Thus, for example, $I_i = 1$ implies that item i is included in the current (partial) solution, whereas $T_t = 0$ implies that transaction t is *not* covered by it. Table 2 lists some of the most common constraints. The *coverage* constraint essentially models a dataset query and ensures that if the item variable assignment corresponds to an itemset p , only those transaction variables that correspond to indices of transactions where p occurs, are assigned value 1. Other constraints allow users to remove uninteresting solutions, e.g., redundant non-*closed* itemsets. Most solvers provide facilities for enumerating all solutions in sequence, i.e., to enumerate all patterns.

In contrast to hard constraints, *quality measures* are used to describe soft user preferences with respect to interestingness of patterns. Common quality measures concern frequency, e.g., $\varphi \equiv \text{freq}$, discriminativity in a labeled dataset, e.g., *purity* $\varphi(p) = \max \{ \text{freq}^+(p), \text{freq}^-(p) \} / \text{freq}(p)$, and others.

4.2 WeightGen

WEIGHTGEN (Chakraborty et al. 2014) is an algorithm for approximate weighted sampling of satisfying assignments (solutions) of a Boolean formula that only requires access to an efficient constraint oracle that enumerates the solutions, e.g., a SAT solver. The core idea consists in partitioning the solution space into a number of “cells” and sampling a solution from a random cell. Partitioning with desired properties is obtained via augmenting the original problem with random XOR constraints. Theoretical guarantees stem from the properties of uniformly random XOR constraints. The sequel follows Sections 3–4 in Chakraborty et al. (2014).

4.2.1 Problem statement and guarantees

Formally, let \mathbf{F} denote a Boolean formula; F a satisfying variable assignment of \mathbf{F} ; M the total number of variables; $w(\cdot)$ a black-box weight function that for each F returns a number in $(0, 1]$; and w_{\min} (resp. w_{\max}) the minimal (resp. maximal) weight over all satisfying assignments of \mathbf{F} . The weight function induces the probability distribution over satisfying assignments of \mathbf{F} , where $P_w(F) = w(F) / \sum w(F')$. Quantity $r = w_{\max} / w_{\min}$ is the (possibly unknown) *tilt* of the distribution induced by $w(\cdot)$.

Given a user-provided upper bound on tilt $\hat{r} \geq r$ and a desired sampling error tolerance $\kappa \in (0, 1)$ (the lower κ , the tighter the bounds on the sampling error), WEIGHTGEN generates a random solution F . Performance guarantees concern both accuracy and efficiency of the algorithm and depend on the parameters and the total number of variables M ; see Sect. 5 for details.

4.2.2 Algorithm

Recall that the core idea that underlies sampling with guarantees is partitioning the overall solution space into a number of random cells by adding random XOR constraints. WEIGHTGEN proceeds in two phases: (1) the estimation phase and (2) the sampling phase. The goal of the *estimation phase* is to estimate the number of XOR constraints necessary to obtain a “small” cell, where the required cell weight is determined by the desired sampling error tolerance.

The *sampling phase* starts with applying the estimated number of XOR constraints. If it obtains a cell whose total weight lies within a certain range, which depends on κ , a solution is sampled exactly from all solutions in the cell; otherwise, it adds a new random XOR constraint. However, the number of XOR constraints that can be added is limited. If the algorithm cannot obtain a suitable cell, it indicates failure and returns no sample.

Both phases make use of a *bounded* oracle that terminates as soon as the total weight of enumerated solutions exceeds a predefined number. It enumerates solutions of the original problem \mathbf{F} augmented with the XOR constraints. An individual XOR constraint over variables \mathbf{X} has the form $\bigotimes b_i \cdot X_i = b_0$, where $b_{0|i} \in \{0, 1\}$. The coefficients b_i determine the variables involved in the constraint, whereas the *parity bit* b_0 determines whether an even or an odd number of variables must be set to 1. Together, m XOR constraints identify one cell belonging to a partitioning of the overall solution space into 2^m cells.

The core operation of WEIGHTGEN involves drawing coefficients uniformly at random, which induces a random partitioning of the solution space that satisfies the *3-wise independence property*, i.e., knowing the cells for two arbitrary assignments does not provide any information about the cell for a third assignment (Gomes et al. 2007b). This ensures desired statistical properties of random partitions, required for the theoretical guarantees. The reader interested in further technical details should consult “Appendix” section and Chakraborty et al. (2014).

5 Flexics: flexible pattern sampler with guarantees

In this paper, we propose FLEXICS, a pattern sampler that uses WEIGHTGEN as the umbrella sampling procedure. To this end, we (1) extend it to CSPs with binary variables, a class of problems that is more general than SAT and that includes pattern mining as described in Sect. 4; (2) augment existing pattern mining algorithms for use with WEIGHTGEN; and (3) investigate the properties of pattern quality measures in the context of WEIGHTGEN’s requirements.

WEIGHTGEN was originally presented as an algorithm to sample solutions of the SAT problem. Pattern mining problems cannot be efficiently tackled by pure Boolean solvers due to the prominence of cardinality constraints (e.g., *minfreq*). However, we observe that the core sampling procedure is applicable to any CSP with binary variables, as its solution space can be partitioned with XOR constraints in the required manner.

Based on this insight, we present two variants of FLEXICS that differ in their oracles. Each oracle is essentially a pattern mining algorithm extended to support XOR constraints along with common constraints on patterns. The first one, dubbed GFLEXICS, builds upon the generic formalization and solving techniques described in Sect. 4 and thus supports a wide range of constraints. Owing to the properties of the *coverage* constraint, XOR constraints only need to involve item variables,¹ which makes them relatively short, mitigating the computational overhead. Moreover, this perspective helps us design the second approach, dubbed EFLEXICS, which uses an extension of ECLAT (Zaki et al. 1997), a well-known mining algorithm, as an oracle. It is tailored for a single task (frequent itemset mining, i.e., it only supports the *minfreq* constraint), but is capable of handling larger datasets. We describe each oracle in detail in the following subsections.

¹ In other words, item variables I are the *independent support* of a pattern mining CSP.

Given a dataset \mathcal{D} , constraints \mathcal{C} , a quality measure φ , and the error tolerance parameter $\kappa \in (0, 1)$, FLEXICS first constructs a CSP corresponding to the task of mining patterns satisfying \mathcal{C} from \mathcal{D} . It then determines parameters for the sampling procedure, including the appropriate number of XOR constraints, and starts generating samples. To this end, it uses one of the two proposed oracles to enumerate patterns that satisfy \mathcal{C} and random XOR constraints. Both variants of FLEXICS support sampling from black-box distributions derived from quality measures and, most importantly, preserve the theoretical guarantees of WEIGHTGEN.²

Theorem 1 *The probability that FLEXICS samples a random pattern p that satisfies constraints \mathcal{C} from a dataset \mathcal{D} , lies within a bounded range determined by the quality of the pattern $\varphi(p)$ and κ :*

$$\frac{\varphi(p)}{Z_\varphi} \times \frac{1}{1 + \varepsilon(\kappa)} \leq P(\text{FLEXICS}(\mathcal{D}, \mathcal{C}, \varphi; \kappa) = p) \leq \frac{\varphi(p)}{Z_\varphi} \times (1 + \varepsilon(\kappa))$$

Proof Theorem 3 of Chakraborty et al. (2014) states:

$$P_w(F) / (1 + \varepsilon(\kappa)) \leq \hat{P}_F \leq P_w(F) \times (1 + \varepsilon(\kappa))$$

where \hat{P}_F denotes the probability that WEIGHTGEN called with parameters \hat{r} and κ samples the solution F , $P_w(F) \propto w(F)$ denotes the target probability of F , and $\varepsilon(\kappa) = (1 + \kappa)(2.36 + 0.51/(1 - \kappa)^2) - 1$ denotes sampling error derived from κ .

For technical purposes, we introduce the notion of the *weight* of a pattern as its quality scaled to the range $(0, 1]$, i.e., $w_\varphi(p) = \varphi(p)/C$, where C is an arbitrary constant such that $C \geq \max_{p \in \mathcal{L}} \varphi(p)$. The proof follows from Theorem 3 of

Chakraborty et al. (2014) and the observation that $\text{FLEXICS}(\mathcal{D}, \mathcal{C}, \varphi; \kappa)$ is equivalent to $\text{WEIGHTGEN}(\text{CSP}(\mathcal{D}, \mathcal{C}), w_\varphi; \kappa)$. The estimation phase effectively corrects for potential discrepancy between C and Z_φ . \square

Furthermore, Theorem 4 of Chakraborty et al. (2014), provides *efficiency guarantees*: the number of calls to the oracle is linear in \hat{r} and polynomial in M and $1/\varepsilon(\kappa)$. The assumption that the tilt is *bounded from above* by a reasonably low number is the only assumption regarding a (black-box) weight function. Moreover, it only affects the efficiency of the algorithm, but not its accuracy.

Thus, using a quality measure with FLEXICS requires knowledge of two properties: scaling constant C and tilt bound \hat{r} . In practice, both are fairly easy to come up with for a variety of measures. For example, for *freq* and *purity*, $C = |\mathcal{D}|$, $\hat{r} = \theta^{-1}$ and $C = 1$, $\hat{r} = 2$ respectively; see Sect. 6 for another example.

5.1 GFlexics: generic pattern sampler

The first variant relies on CP4IM (Guns et al. 2011), a constraint programming-based mining system. A wide range of constraints supported by CP4IM are automatically

² Theorem 1 corresponds to and follows from Theorem 3 of Chakraborty et al. (2014).

$x_1 \otimes x_5 = 1$	$1 \ 0 \ 0 \ 0 \ 1 \mid 1$	$\rightarrow 1 \ 0 \ 0 \ 0 \ 1 \mid 1$	$1 \ 0 \ 0 \ 0 \ 1 \mid 1$	$\rightarrow \begin{matrix} \downarrow & & & \downarrow \\ 0 & 0 & 0 & 0 & 0 \end{matrix} \mid \begin{matrix} 1 \\ 0 \end{matrix}$
$x_2 \otimes x_3 \otimes x_4 \otimes x_5 = 0$	$0 \ 1 \ 1 \ 1 \ 1 \mid 0$	$\rightarrow 0 \ 1 \ 0 \ 0 \ 0 \mid 0$	$0 \ 0 \ 0 \ 1 \ 1 \mid 1$	$0 \ 0 \ 0 \ 1 \ 0 \mid 0$
$x_1 \otimes x_2 \otimes x_3 \otimes x_5 = 0$	$1 \ 1 \ 1 \ 0 \ 1 \mid 0$	$\rightarrow 0 \ 0 \ 1 \ 0 \ 0 \mid 1$	$0 \ 0 \ 0 \ 0 \ 0 \mid 0$	$0 \ 0 \ 0 \ 0 \ 0 \mid 0$
$x_2 \otimes x_4 \otimes x_5 = 1$	$0 \ 1 \ 0 \ 1 \ 1 \mid 1$	$0 \ 0 \ 0 \ 1 \ 1 \mid 1$	$0 \ 0 \ 0 \ 0 \ 0 \mid 0$	$0 \ 0 \ 0 \ 0 \ 0 \mid 0$
(a)	(b)	(c)	(d)	(e)

Fig. 1 Propagating a system of XOR constraints using Gaussian elimination in \mathbb{F}_2 . **a** Random XOR constraints. **b** Initial constraint matrix. **c** Echelonized matrix: assignments $x_2 = 0$ and $x_3 = 1$ are derived. **d** Updated matrix (rows 2 and 4 are swapped). **e** If x_1 and x_5 are set to 1 (e.g., by search), the system is unsatisfiable

supported by the sampler and can be freely combined with various quality measures.

In order to turn CP4IM into a suitable bounded oracle, we need to extend it with an efficient propagator for XOR constraints. This propagator is based on the process of *Gaussian elimination* (Gomes et al. 2007a), a classical algorithm for solving systems of linear equations. Each XOR constraint can be viewed as a linear equality over the field \mathbb{F}_2 of two elements, 0 and 1, and all coefficients form a binary matrix (Fig. 1b). At each step, the matrix is updated with the latest variable assignments and transformed to *row echelon form*, where all ones are on or above the main diagonal and all non-zero rows are above any rows of all zeroes (Fig. 1c). During echelonization, two situations enable propagation. If a row becomes empty while its right hand side is equal to 1, the system is unsatisfiable and the current search branch terminates (Fig. 1e). If a row contains only one free variable, it is assigned the right hand side of the row (Fig. 1c).

Gaussian elimination in \mathbb{F}_2 can be performed very efficiently, because no division is necessary (all coefficients are 1), and subtraction and addition are equivalent operations. For a system of k XOR constraints over n variables, the total time complexity of Gaussian elimination is $\mathcal{O}(k^2n)$.

5.2 EFlexics: efficient pattern sampler

Generic constraint solvers currently cannot compete with the efficiency and scalability of specialized mining algorithms. In order to develop a less flexible, yet more efficient version of our sampler, we extend the well-known ECLAT algorithm to handle XOR constraints. Thus, EFLXICS is tailored for frequent itemset sampling and uses ECLATXOR (Algorithm 1) as an oracle.

Algorithm 1 shows the pseudocode of the extended ECLAT. The algorithm relies on the *vertical* data representation, i.e., for each candidate item, it stores a set of indices of transactions (TIDs), in which this item occurs (Line 4). ECLAT starts with determining frequent items and ordering them, typically by frequency ascending. It explores the search space in a depth-first manner, where each branch corresponds to (ordered) itemsets that share a prefix.

The core operation is referred to as *processing an equivalence class* of itemsets (EQCLASS). For each prefix, ECLAT maintains a set of candidate suffixes, i.e., items that follow the last item of the prefix in the item order and are frequent. The frequency

Algorithm 1 ECLAT augmented with XOR constraint propagation (Lines 16–22)**Input:** Dataset \mathcal{D} over items \mathcal{I} , min.freq θ , XOR matrix M **Assumes:** Item order $>_{\mathcal{I}}$ by frequency ascending

```

1: function ECLATXOR( $\mathcal{D}, \theta, M$ )
     $\triangleright$  Mine all frequent patterns that satisfy XOR constraints encoded by  $M$ 
2:   Frequent items  $FI = \emptyset$ 
3:   for item  $i \in \mathcal{I}$  do
4:      $TID_i = \{\text{transaction index } t \in \mathcal{T} \mid \mathcal{D}_{ti} = 1\}$ 
5:     if  $|TID_i| \geq \theta$  then  $\triangleright$  Item is frequent
6:        $FI \xleftarrow{\text{Add}} (i, TID_i)$ 
7:   SORT( $FI, >_{\mathcal{I}}$ )
8:   for  $i \in FI$  do
9:     Candidate suffixes  $CS = \{i' \in FI \setminus i \mid i' >_{\mathcal{I}} i\}$ 
10:    EQCLASS( $\{i\}, CS, M$ )
11: function EQCLASS(Prefix  $P$ , cand.suffixes  $CS \neq \emptyset, M$ )
     $\triangleright$  Mine all patterns that start with  $P$ 
12:   if CHECKCONSTRAINTS( $P, M$ ) then
13:     return  $P$   $\triangleright$  Return prefix, if it satisfies XORs
14:   for candidate suffix  $s \in CS$  do
15:      $P' = P \cup s$ ; frequent suffixes  $FS =$ 
        $\{f \in CS \setminus s \mid f >_{\mathcal{I}} s \wedge |f.TID \cap s.TID| \geq \theta\}$ 
        $\triangleright$  Propagate XOR constraints
16:      $U_1 = \{s\}, U_0 = CS \setminus FS$   $\triangleright$  Variable updates
17:      $M' = \text{UPDATEANDECHELONIZE}(M, U_1, U_0)$ 
18:      $(A_1, A_0) = \text{PROPAGATE}(M')$   $\triangleright$  Item variables
        $\triangleright$  that were assigned value 1 or 0 by propagation
19:      $FS' = FS \setminus (A_1 \cup A_0)$ 
20:     if  $A_1 \neq \emptyset$  then  $\triangleright$  If prefix was extended,
        $\triangleright$  update TIDs and check support
21:        $P' \leftarrow P' \cup A_1, \Delta_{TID} = \bigcap_{f \in A_1} f.TID$ 
22:        $FS' \leftarrow \{f' \in FS' : |f'.TID \cap \Delta_{TID}| \geq \theta\}$ 
23:   if  $|P'.TID| \geq \theta \wedge FS' \neq \emptyset$  then
24:     EQCLASS( $P', FS', M'$ )

```

of a candidate suffix, given the prefix, is computed by intersecting its TID with the TID of the prefix (Lines 9, 15, and 22).

We extend ECLAT with XOR constraint handling (Lines 16–22). Variable updates stem from ECLAT extending the prefix and removing infrequent suffixes (Line 16). XOR propagation can result in extending the prefix or removing candidate suffixes as well (Line 19). Furthermore, if the prefix has been extended, TIDs of candidate suffixes need to be updated, with some of them possibly becoming infrequent, leading to further propagation (Lines 19–22). If the prefix becomes infrequent, the search branch terminates.

Fixed variable-order search, like ECLAT, is an advantageous case for Gaussian elimination (Soos 2010): non-zero elements are restricted to the right region of the matrix, hence Gaussian elimination only needs to consider a contiguous, progressively shrinking subset of columns. Total memory overhead of ECLATXOR compared to plain ECLAT is $\mathcal{O}(d \times |\mathcal{F}| \times N_{XOR} + \text{pivot} \times r)$, where d denotes maximal search depth, $|\mathcal{F}|$ the number of frequent singletons (columns of a matrix), and N_{XOR} the number

of XOR constraints (rows of a matrix). The first term refers to a set of XOR matrices in unexplored search branches, whereas the second term refers to storing itemsets in a cell (Line 17 in Algorithm 2 in “Appendix” section).

6 Pattern set sampling

We highlight the flexibility of FLEXICS by introducing and tackling the novel task of *sampling sets of patterns*. For the purposes of sampling, a set of patterns is essentially treated as a composite pattern. Typically, constituent patterns are required to be different from each other. The quality (and hence, the sampling probability) of a pattern set depends on collective properties of constituent patterns. These characteristics, coupled with the immense size of the pattern set search space, make sampling even more challenging.

To develop a sampler, we extend GFLEXICS with the CSP-formulation of the k -pattern set mining task (Guns et al. 2013), which in turn builds upon the formulation of the itemset mining task described in Sect. 4. Recall that a CSP is defined by a set of variables and constraints over these variables. Each constituent pattern is modeled with distinct item and transaction variables, i.e., I_{ik} and T_{tk} for the k th pattern p_k . Note that this increases the length of XOR constraints, which poses an additional challenge from the sampling perspective.

Any single-pattern constraint can be enforced for a constituent pattern, e.g., *minfreq* (θ), *closed*, or *minlen* (λ). A common pattern set-specific constraint is *no overlap*, which enforces that neither the itemsets (1), nor the sets of transactions that they cover (2) overlap:

$$(1) \forall i \in \mathcal{I} \quad \sum I_{ik} \leq 1 \quad (2) \forall t \in \mathcal{T} \quad \sum T_{tk} \leq 1$$

Furthermore, there is typically a symmetry-breaking constraint that requires that the set of transaction indices of p_i lexicographically precedes those of $\{p_j \mid j > i\}$. This approach allows modeling a wide range of pattern set sampling tasks, e.g., sampling k -term DNFs, conceptual clusterings, redescrptions, and others. In this paper, we use the problem of *tiling datasets* (Geerts et al. 2004) as an example.

The main aim of tiling is to cover a large number of 1s in a binary 0/1 dataset with a given number of patterns. Thus, a *tiling* is essentially a set of itemsets that together describe as many item occurrences as possible. Without loss of generality, we describe the task of sampling non-overlapping 2-tilings ($k = 2$). Let p_1 and p_2 denote the constituent patterns of a 2-tiling. The quality of a tiling is equal to its *area*, i.e., the number of 1s that it covers:

$$area(\{p_1, p_2\}) = (freq(p_1) \times |p_1| + freq(p_2) \times |p_2|)$$

The scaling constant for *area* is $C = \sum \mathcal{D}_{ti}$, i.e., the total number of 1s in the dataset. The tilt bound is estimated as $\hat{r} = \sum \mathcal{D}_{ti} / (2 \times (|\mathcal{D}| \times \theta) \times \lambda)$, where the denominator is the smallest possible area of a 2-tiling given the constraints.

Table 3 Combinations of two constraint sets and three quality measures yield six experimental settings per dataset for sampling accuracy experiments; see Sect. 4 for definitions

	Constraints \mathcal{C}	Itemsets per dataset	Quality measure φ	Tilt bound \hat{r}
F	$\min\mathbf{Freq}(\theta)$	$\sim 60\,000$	$\text{uniform } (\varphi \equiv 1)$	1
FCL	$\min\mathbf{Freq}(\theta) \wedge$	$\geq 15\,000$	purity	2
	$\mathbf{Closed} \wedge \min\mathbf{Len}(\lambda)$		freq	θ^{-1}

7 Experiments

The experimental evaluation focuses on accuracy, scalability, and flexibility of the proposed sampler. The research questions are as follows:

- Q1 *How close is the empirical sampling distribution to the target distribution?*
- Q2 *How does FLEXICS compare to the specialized alternatives?*
- Q3 *Does FLEXICS scale to large datasets?*
- Q4 *How flexible is FLEXICS, i.e., can it be used for new pattern sampling tasks?*

The implementations of GFLEXICS and EFLEXICS³ are based on CP4IM⁴ and a custom implementation of ECLAT respectively. Both are augmented with a propagator for a system of XOR constraints based on the implementation of Gaussian elimination in the M4RI library⁵ (Albrecht and Bard 2012). All experiments were run on a Linux machine with an Intel Xeon CPU@3.2 GHz and 32 Gb of RAM.

7.1 Q1: sampling accuracy

We study the sampling accuracy of GFLEXICS in settings with tight constraints, which yield a relatively low number of solutions. This allows us to compute the exact statistical distance between the empirical sampling distribution and the target distribution. We investigate settings with various quality measures and constraint sets as well as the effect of the tolerance parameter κ .

We select several datasets from the CP4IM repository⁶ in the following way. For each dataset, we construct two constraint sets (see Table 3). We choose a value of θ such that there are approximately 60,000 frequent patterns. Given θ , we choose a value of $\lambda \geq 2$ such that there are at least 15,000 closed patterns that satisfy the *minlen* constraint. In order to obtain sufficiently challenging sampling tasks, we omit the datasets where the latter condition does not hold (i.e., there are too few closed “long” patterns). Combining two constraint sets with three quality measures yields six experimental settings per dataset. Table 5 shows dataset statistics and parameter values. For each $\kappa \in \{0.1, 0.5, 0.9\}$, we request 900,000 samples.

³ Available at <https://bitbucket.org/wxd/flexics>.

⁴ <https://dtai.cs.kuleuven.be/CP4IM>.

⁵ <https://bitbucket.org/malb/m4ri/>.

⁶ Source: <https://dtai.cs.kuleuven.be/CP4IM/datasets/>.

Table 4 Sampling accuracy of FLEXICS (here GFLEXICS) is consistently high across quality measures, constraint sets [*minFreq* (0.09) vs. *minFreq* (0.09) \wedge *Closed* \wedge *minLen* (7)], and error tolerance κ

κ	vote dataset, JS-divergence from target					
	Uniform ($\hat{r} = 1$)		Purity ($\hat{r} = 2$)		Frequency ($\hat{r} = 11$)	
	F	FCL	F	FCL	F	FCL
0.9	0.013	0.004	0.013	0.004	0.013	0.004
0.5	0.013	0.004	0.013	0.004	0.013	0.004
0.1	0.013	0.004	0.013	0.004	0.013	0.004
Ideal sampler	0.013	0.004	0.013	0.004	0.013	0.004

JS-divergence is impressively low, equivalent to that of the ideal sampler

Let T denote the set of all itemsets that satisfy the constraints, E denote the multiset of all samples, and $\mathbf{1}_S$ its multiplicity function. For a given quality measure φ , target and empirical probabilities of sampling an itemset p are respectively defined as $P_T(p) = \varphi(p) / \sum_{p' \in T} \varphi(p')$ and $P_E(p) = \mathbf{1}_E(p) / |E|$. We use *Jensen–Shannon (JS) divergence* to quantify the statistical distance between P_T and P_E . Let $D_{KL}(P_1 \| P_2)$ denote the well-known *Kullback–Leibler divergence* between distributions P_1 and P_2 . JS-divergence D_{JS} is defined as follows:

$$D_{JS}(P_T \| P_E) = 0.5 \times (D_{KL}(P_T \| P_M) + D_{KL}(P_E \| P_M))$$

$$\text{where } P_M = 0.5 \times (P_T + P_E)$$

JS-divergence ranges from 0 to 1 and, unlike KL-divergence, does not require that $P_T(p) > 0 \Rightarrow P_E(p) > 0$, i.e., that each solution is sampled at least once, which does not always hold in sampling experiments. We compare D_{JS} attained with our sampler with that of the ideal sampler, which materializes all itemsets satisfying the constraints, computes their qualities, and uses these to sample directly from the target distribution.

7.1.1 A characteristic experiment in detail

Our experiments show that results are consistent across various datasets. Therefore, we first study the results on the *vote* dataset in detail. Table 4 shows that the theoretical error tolerance parameter κ has no considerable effect on practical performance of the algorithm, except for runtime, which we evaluate in subsequent experiments. One possible explanation is the high quality of the output of the estimation phase, which thus alleviates theoretical risks that have to be accounted for in the general case (see below for a numerical characterization). Hence, in the following experiments we use $\kappa = 0.9$ unless noted otherwise.

JS-divergences for different quality measures and constraint sets are impressively low, equivalent to the highest possible sampling accuracy attainable with the ideal sampler. Figure 2 illustrates this for *minfreq* (0.09) \wedge *closed* \wedge *minlen* (7), $\varphi = \text{freq}$, and $\kappa = 0.9$ ($D_{JS} = 0.004$): the sampling frequency of an average itemset is close to the target probability. For at least 90% of patterns, the sampling error does not exceed a factor of 2.

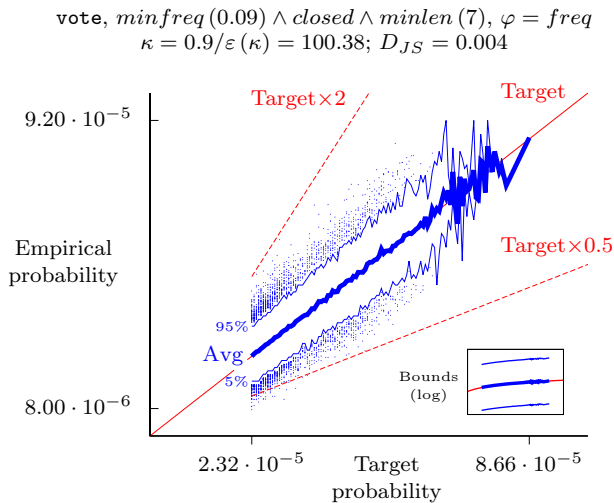


Fig. 2 Empirical sampling frequencies of itemsets that share the same target probability, i.e., have the same quality. On average, frequencies are close to the target probabilities. 90% of frequencies are well within a factor 2 from the target, which is considerably lower than the theoretical factor of 100.38. The *dots* show the tails of the empirical probability distribution for a given target probability. The *lower right box* shows theoretical bounds and empirical frequencies on the logscale

Table 5 Dataset statistics and parameter values and results of sampling accuracy experiments

	$ \mathcal{D} $	$ \mathcal{I} $	Density (%)	θ	λ	JS-divergence, $\kappa = 0.9$					
						Uniform		Purity		Frequency	
						F	FCL	F	FCL	F	FCL
german	1000	112	34	0.35 (349)	2	0.012	0.003	0.013	0.003	0.013	0.003
heart	296	95	47	0.43 (127)	2	0.012	0.003	0.012	0.003	0.012	0.003
hepatitis	137	68	50	0.39 (53)	5	0.013	0.004	0.014	0.004	0.013	0.004
kr-vs-kp	3196	74	49	0.69 (2190)	6	0.013	0.005	0.013	0.005	0.013	0.005
primary	336	31	48	0.09 (30)	7	0.013	0.004	0.013	0.004	0.013	0.004
splice	3190	287	21	0.04 (122)	3	–	–	–	–	–	–
vote	435	48	33	0.09 (40)	7	0.013	0.004	0.013	0.004	0.013	0.004

Even with high error tolerance $\kappa = 0.9$, JS-divergence of FLEXICS (here GFLEXICS) is consistently low across datasets, quality measures, and constraint sets. On the *splice* dataset, GFLEXICS generates less than 900,000 samples before the timeout; see also Table 7

Table 5 shows that similar conclusions hold for several other datasets. Over all experimental settings, the error of the estimation of the total weight of all solutions, which is used to derive the number of XOR constraints for the sampling phase, never exceeds 10%, whereas the bounds assume the error of 45–80%. This helps explain why practical errors are considerably lower than theoretical bounds.

Table 6 The accuracy of FLEXICS (here GFLEXICS) is consistent across settings

	Uniform		Frequency			FCL		
	F		F					
	GF	ACFI	GF	TS $\sim freq$	TS $\sim freq^4$	GF	TS $\sim freq$	TS $\sim freq^4$
JS-divergence (for TS, acceptance rate)								
german	0.01	0.01	0.01	$-(9 \times 10^{-8})$	$-(0.02)$	0.00	$-(5 \times 10^{-8})$	$-(0.06)$
heart	0.01	0.01	0.01	$-(4 \times 10^{-10})$	$-(0)$	0.00	$-(0)$	$-(3 \times 10^{-3})$
hepatitis	0.01	0.01	0.01	$-(2 \times 10^{-6})$	$-(0.01)$	0.00	$-(1 \times 10^{-6})$	$-(0.01)$
kr-vs-kp	0.01	0.01	0.01	$-(7 \times 10^{-7})$	$-(0.01)$	0.01	$-(4 \times 10^{-7})$	$-(4 \times 10^{-3})$
primary	0.01	0.01	0.01	0.01 (0.30)	0.40 (0.99)	0.01	0.01 (0.13)	0.27 (0.10)
splice	–	0.01	–	$-(0)$	$-(0)$	–	$-(0)$	$-(0)$
vote	0.01	0.01	0.01	0.01 (0.13)	0.23 (0.94)	0.00	0.01 (0.05)	0.14 (0.22)

In uniform frequent itemset sampling, performance of FLEXICS as well as of ACFI is equivalent to that of the ideal sampler (not shown). In frequency-weighted sampling, it is comparable to the exact two-step sampler (TS $\sim freq$) with rejection. However, the latter suffers from low acceptance rates, which, for settings marked with ‘–’, is not improved by increasing bias (TS $\sim freq^4$). On *splice*, neither TS nor FLEXICS generate 900,000 samples before the timeout; see also Table 7

In line with theoretical expectations (see Sect. 5), the *splice* dataset proves the most challenging due to the large number of items (variables in XOR constraints). As a result, GFLEXICS does not generate the requested number of samples within the 24 h timeout. We study the runtime in the following experiment.

7.2 Q2: comparison with alternative pattern samplers

We compare FLEXICS to ACFI (Boley and Grosskreutz 2009) and TS (Boley et al. 2012), alternative samplers⁷ described in Sect. 3, in the settings that they are tailored for. ACFI only supports the setting with a single $\minfreq(\theta)$ constraint and $\varphi = \text{uniform}$. It is run with a burn-in of 100,000 steps and uses a built-in heuristic to determine the number of steps between consecutive samples. TS is evaluated in the setting with $\varphi = freq$ and both constraint sets from the previous experiments. It samples from two of the distributions it supports, $freq$ and $freq^4$; samples that do not satisfy the constraints are rejected. Both samplers are requested to generate 900,000 samples and are allowed to run up to 24 h. Datasets and parameters are identical to the previous experiments.

Table 6 shows the accuracy of the samplers. The performance of FLEXICS is on par with specialized samplers. That is, in uniform frequent itemset sampling, the accuracy of both FLEXICS and ACFI is equivalent to that of the ideal sampler and can therefore not be improved. When sampling proportional to frequency, it is equivalent to the accuracy of the exact two-step sampler TS $\sim freq$. However, the latter does not directly take constraints into account, which poses considerable problems on most

⁷ The code was provided by their respective authors. We also obtained the “unmaintained” code for the *uniform* LRW sampler (personal communication), but were unable to make it run on our machines. The code for the FCA sampler was not available (personal communication).

Table 7 Runtime in milliseconds required to sample a frequent itemset, including pre-processing, i.e., estimation or burn-in, amortized over 1000 samples

	$\varphi = \text{uniform}, \mathcal{C} = \text{F}$			$\varphi = \text{freq}, \mathcal{C} = \text{F}$		
	GFLEXICS	EFLEXICS	ACFI	GFLEXICS	EFLEXICS	TS $\sim \text{freq}$
german	110	25	39	133	34	58,540
heart	60	45	24	73	44	–
hepatitis	23	33	11	30	45	2632
kr-vs-kp	59	9	6	59	10	8731
primary	10	10	4	27	25	0.10
splice	170,360	1376	580	–	1095	–
vote	25	19	8	46	28	0.03

Both variants of FLEXICS are suitable for anytime exploration, although slower than the specialized samplers. The two-step sampler is the fastest in the task it is tailored for, but fails in the settings with tighter constraints. EFLEXICS provides runtime benefits compared to GFLEXICS

datasets. For example, for the `heart` dataset, TS fails to generate a single accepted sample, despite generating 2 billion unconstrained candidates. This issue is not solved by increasing the bias towards more frequent itemsets by sampling proportional to freq^4 . Furthermore, this would substantially decrease accuracy, as seen in `primary` and `vote`.

Table 7 shows the runtimes for frequent itemset sampling (i.e., only the *minfreq* constraint). In most settings, EFLEXICS provides runtime benefits over GFLEXICS. The `splice` dataset is the most challenging due to the large number of items; it highlights the importance of an efficient constraint oracle. Accordingly, the specialized sampler ACFI is from 6 to 22 ms faster than a faster variant of FLEXICS in uniform sampling (excluding `splice`). In frequency-weighted sampling, FLEXICS is considerably faster in the settings with tighter constraints, where the two-step sampler is slow to generate accepted samples. This illustrates the overhead as well as the benefits of the flexibility of the proposed approach. Furthermore, in these settings, there are at most 66,000 patterns, which is too low to suggest the need for pattern sampling (recall that the primary goal of these experiments was to evaluate and compare sampling accuracy) and does not allow for the overhead amortization. We therefore tackle settings with a substantially larger number of patterns in the following experiments.

7.3 Q3: scalability

To study scalability of the proposed sampler, we compare its runtime costs with those required to construct an ideal sampler with LCM,⁸ an efficient frequent itemset miner (Uno et al. 2005). To this end, we estimate the costs of completing the following scenario: pre-processing (estimation or counting), followed by sampling 100 itemsets

⁸ <http://research.nii.ac.jp/~uno/codes.htm>, ver. 3.

Table 8 EFLEXICS generates individual samples considerably faster than LCM, although it is slower in counting

	\mathcal{D}	\mathcal{I}	Density (%)	θ	Itemsets (bln.)	Counting (min)		Sampling (s)	
						LCM	EFLEXICS	LCM	EFLEXICS
accidents	340,183	469	7.21	0.009	5.37	1.55	6.48	33.77	10.30
connect	67,557	130	33.08	0.178	16.88	0.01	0.38	59.00	0.37
kosarak	990,002	41,271	0.02	0.042	10.93	4.87	456.30	73.04	294.89
pumsb	49,046	7117	1.04	0.145	1.11	0.09	1.19	18.14	0.75

The *kosarak* dataset poses a significant challenge to EFLEXICS due to its number of items and sparsity that complicate the propagation of XOR constraints

in two batches of 50. We use non-synthetic datasets from the FIMI repository,⁹ which have fewer than one billion transactions and select θ such that there are more than one billion frequent itemsets (see Table 8).

7.3.1 A characteristic experiment in detail

We use the *accidents* dataset (469 items, 340,183 transactions) and $\theta = 0.009$ (3000 transactions), which results in a staggering number of 5.37 billion frequent itemsets. We run WEIGHTGEN with values of $\kappa \in \{0.1, 0.5, 0.9\}$. (Note that the estimation phase is identical for all three cases.) The baseline sampler is constructed as follows. LCM is first run in counting mode, which only returns the total number of itemsets. Then, for each batch, 50 random line numbers are drawn, and the corresponding itemsets are printed while LCM is enumerating the solutions.¹⁰ The latter phase is implemented with the standard Unix utility ‘awk’.

Figure 3 illustrates the results. The counting mode of LCM is roughly 4.5 min faster than the estimation phase of EFLEXICS. Generating samples from the output of LCM, on the other hand, is considerably slower: it takes approximately 35 s to sample one itemset, whereas EFLEXICS takes from 10 to 27 s per sample, depending on error tolerance κ . As a result, EFLEXICS samples two batches faster than LCM regardless of its parameter values. Moreover, with $\kappa = 0.9$ it samples all 100 itemsets even before the first batch is returned by LCM.

Thus, the proposed sampler outperforms a sampler derived from an efficient itemset miner, even though the experimental setup favors the latter. First, non-uniform weighted sampling would require more advanced computations with itemsets, which would increase the costs of both counting and sampling with LCM. Second, EFLEXICS could also benefit from the exact count obtained by LCM and start sampling after 1.5 min. Third, the individual itemsets sampled from the output of an algorithm based on deterministic search are not *exchangeable*. Figure 4 illustrates this: due to LCM’s

⁹ <http://fimi.ua.ac.be/data/>.

¹⁰ Storing all itemsets on disk provides no benefits: it increases the mining runtime to 23 min and results in a file of 215 Gb; simply counting its lines with ‘wc -l’ takes 25 min.

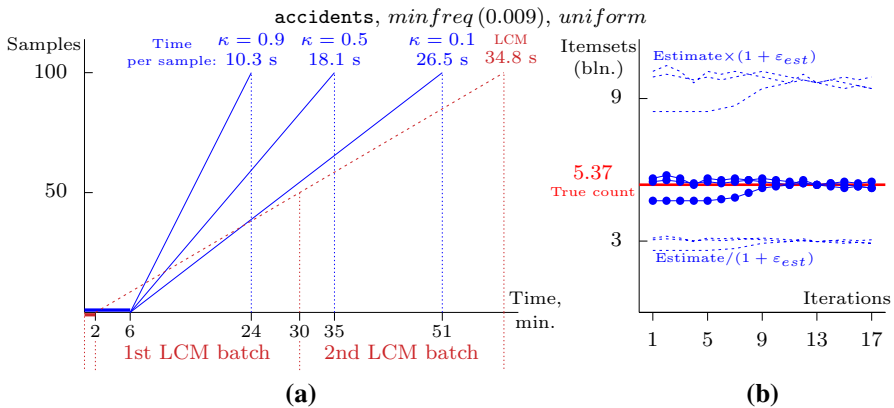


Fig. 3 **a** Sampling runtime comparison. EFlexics generates two batches of 50 samples faster than a sampler derived from LCM, regardless of error tolerance. **b** Estimation accuracy. EFlexics with the uniform quality converges to a high-quality estimate of the total number of itemsets in a small number of iterations (three different random seeds shown). Practical error of the estimation phase is substantially lower than theoretical bounds, which indirectly signals high sampling accuracy

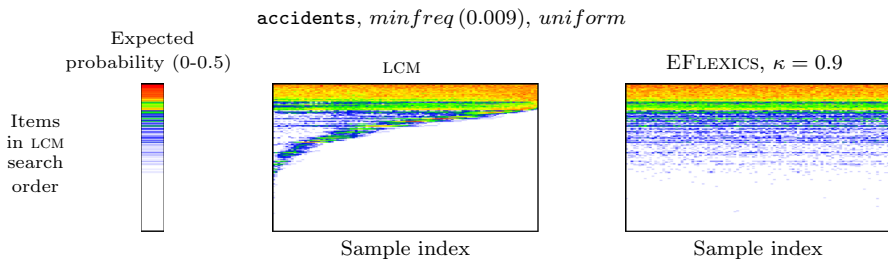


Fig. 4 The probability of observing a given item at a certain position in a batch by EFLEXICS is close to the expected probability of observing this item in a random itemset, which indicates high sampling accuracy. The samples by the LCM-based sampler are not exchangeable, i.e., certain items are under- or oversampled at certain positions in a batch, depending on their position in LCM's search order

search order, certain items only occur at the beginning of batches, while for EFLEXICS, the order within a batch is random.

The accuracy of FLEXICS in this scenario can be evaluated indirectly, by comparing the estimate of the total number of itemsets obtained at the estimation phase with the actual number. The error tolerance of the estimation phase is $\varepsilon_{est} = 0.8$ (see “Appendix” section for details). Figure 3b demonstrates that, in practice, the error is substantially lower than the theoretical bound. Furthermore, 3–9 iterations suffice to obtain an accurate estimate. Similar to previous experiments, accurate input from the estimation phase alleviates theoretical risks and is expected to enable accurate sampling.

Table 8 summarizes the results. On three out of four datasets, LCM is faster in counting itemsets, but considerably slower in generating individual samples, which is even more pronounced on connect and pumsb than on accidents. The results are opposite on the kosarak dataset, which is in line with the theoretical expectations

Table 9 Time required to sample a 2-tiling is approximately 4 s, which is suitable for anytime exploration

	θ	λ	Tilt bound \hat{r}	Tilings (mln.)	Enumeration (min)	Sampling with GFLEXICS	
						Estimation (min)	Per sample (s)
german	0.22	3	25.4	11.2	8.2	12.6	15.3
heart	0.30	5	13.3	2.2	1.0	3.3	3.9
hepatitis	0.26	5	12.4	7.2	1.9	2.6	3.6
kr-vs-kp	0.31	4	13.1	20.3	18.5	3.5	5.1
primary	0.03	5	50.3	24.9	5.5	4.0	4.5
vote	0.10	5	15.3	170.1	37.0	2.9	4.4

Runtime benefits of the sampling procedure are the largest for the settings with the largest tiling counts (kr-vs-kp, primary, and vote)

(see Sect. 5): the large number of items and the sparsity of the dataset sharply increase the costs of XOR constraint propagation. As a result, enumeration with ECLAT within EFLEXICS becomes considerably slower than with LCM (augmenting LCM to handle XOR constraints might provide a solution, but is challenging from an implementation perspective).

7.4 Q4: pattern set sampling

In order to demonstrate the flexibility of our approach and the promised benefits of weighted constrained pattern sampling, i.e., (1) diversity and quality of results, (2) utility of constraints, and (3) the potential for anytime exploration, we here address the problem of sampling non-overlapping 2-tilings as introduced in Sect. 6. We reuse the implementation of GFLEXICS from the itemset sampling experiments, only modifying the declarative specification of the CSP. Likewise, we impose the FCL constraints on constituent patterns.

Table 9 shows parameters and runtimes for sampling 2-tilings proportional to *area*. The time to sample a single 2-tiling is suitable for pattern-based data exploration, where tilings are inspected by a human user, as it exceeds 5s only on the *german* dataset. For several settings, the estimation phase runtime slightly exceeds the runtime of enumerating all solutions. However, for the settings with a large number of pattern sets, which are arguably the primary target of pattern samplers, the opposite is true. For example, in the *vote* experiment with 170 million tilings, the estimation phase runtime only amounts to 8% of the complete enumeration runtime, which demonstrates the benefits of the proposed approach.

The left part of Fig. 5 shows six random 2-tilings sampled from the *vote* dataset. Constraints ensure that the individual tiles comprising each 2-tiling do not overlap, simplifying interpretation. Moreover, the *set* of tilings is diverse, i.e., the tilings are dissimilar to each other. They cover different regions in the data, revealing alternative structural regularities.

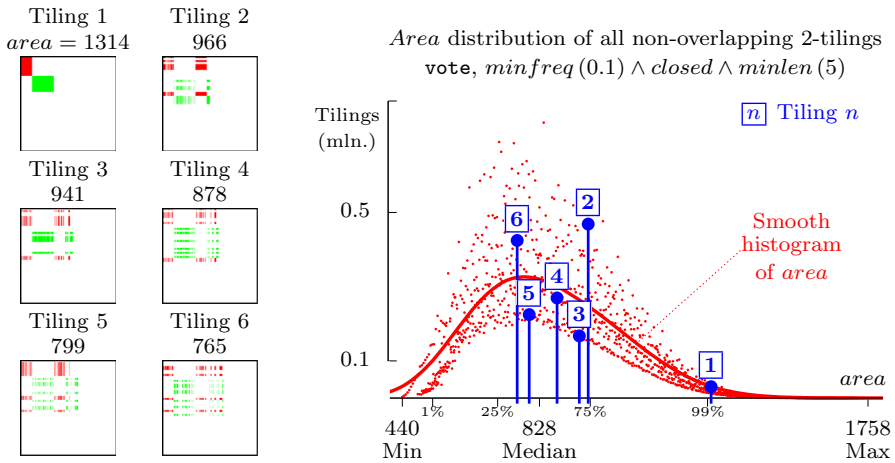


Fig. 5 Left Six 2-tilings sampled consecutively from the *vote* dataset. The tilings are diverse, i.e., cover different regions in the data, a property essential for pattern-based data exploration. Note that while the sampled tilings are fair random draws, the *images* are not random: the tilings were sorted by area descending, and items and transactions were re-arranged so that the cells covered by tilings with larger area are as close to each other as possible. Right Qualities (area) of the samples, indicated by vertical bars, tend towards a dense region between the 25th and the 75th percentile

The right part of Fig. 5 shows the *area* distribution of all 2-tilings that satisfy the constraints, obtained by complete enumeration. Qualities of 5 out of 6 tilings fall in the dense region between the 25th and 75th percentile, indicating high sampling accuracy. This is completely expected from the problem statement. In practice, pattern quality measures, like *area*, are only an approximation of application-specific pattern interestingness, thus diversity of results is a desirable characteristic of a pattern sampler as long as the quality of individual patterns is sufficiently high. To sample patterns from the right tail (i.e., with exceptionally high qualities) more frequently, the sampling task could be changed, e.g., either by choosing another sampling distribution or by enforcing constraints on *area*.

8 Discussion

The experiments demonstrate that FLEXICS delivers the promised benefits: (1) it is flexible in that it supports a wide range of pattern constraints and sampling distributions in itemset mining as well as the novel pattern set sampling task; (2) it is anytime in that the time it takes to generate random patterns is suitable for online data exploration, including the settings with large datasets or large solution spaces; and (3) by virtue of high sampling accuracy in all supported settings, sampled patterns are diverse, i.e., originate from different regions in the solution space. The theoretical guarantees ensure that the empirical observations extend reliably beyond the studied settings. Furthermore, practical accuracy is substantially higher than theory guarantees. The results confirm that pattern mining can benefit from the latest advances in

AI, particularly in weighted constrained sampling for SAT. In this section, we discuss potential applications, advantages, and limitations of the proposed approach.

The primary application of pattern sampling involves showing sampled patterns directly to the user. In exploratory data analysis, the mining task is often ill-defined, i.e., the quality measure and the constraints reflect the application-specific pattern interestingness only approximately (Carvalho et al. 2005). Owing to its flexibility, FLEXICS allows experimenting with various task formulations using the same algorithm. Pattern sampling allows obtaining diverse and representative sets of patterns in an anytime manner. These properties are particularly important in *interactive mining systems*, which aim at returning patterns that are *subjectively* interesting to the current user. Boley et al. (2013) used two-step samplers in such a system, while Dzyuba and van Leeuwen (2017) proposed to learn low-tilt subjective quality measures specifically for sampling with FLEXICS.

Furthermore, the theoretical guarantees enable applications beyond displaying the sampled patterns: FLEXICS can be plugged into algorithms that use patterns as building blocks for pattern-based models, yielding anytime versions thereof with (ϵ, δ) -approximation guarantees of their own derived from FLEXICS' guarantees. Example approaches include community detection with ECLAT (Berlingerio et al. 2013) or outlier detection with two-step sampling (Giacometti and Soulet 2016). The authors note that the formulation of the mining task has a strong influence on the results in the respective applications. FLEXICS allows the algorithm designer to experiment with these choices and thus to obtain variants of these approaches, perhaps with better application performance.

The flexibility also provides algorithmic advantages. In addition to being agnostic of the quality measure φ and the constraint set \mathcal{C} , FLEXICS is also agnostic of the underlying solution space and the oracle, as long as (1) solutions can be encoded with binary variables and (2) the oracle supports XOR constraints. Thus, FLEXICS provides a principled method to convert a pattern enumeration algorithm into a sampling algorithm, which amounts to implementing the mechanism to handle XOR constraints. This allows re-using algorithmic advances in pattern mining for developing pattern samplers, which we accomplished with CP4IM and ECLAT.

Most importantly, FLEXICS' black-box nature simplifies extensions to new pattern languages. For example, possible extensions of GFLEXICS cover a variety of pattern set languages in Guns et al. (2013), e.g., conceptual clustering. EFLEXICS can be extended to sample other binary pattern languages, e.g., association rules (Agrawal et al. 1996) or redescrptions (Ramakrishnan et al. 2004). In contrast, MCMC algorithms, like LRW, are based on local neighborhood enumeration, which is uncommon in traditional pattern mining techniques, and thus require distinctive design and implementation principles for novel problems.

On the other hand, FLEXICS only supports pattern languages that can be compactly represented with binary variables, such as the itemsets and pattern sets studied in this paper. This essentially limits it to propositional discrete (binary, categorical, or discretized numeric) data. While in principle structured pattern languages, e.g., sequences or graphs, could also be modeled using this framework, the number of variables would rise sharply, which would negatively affect performance. Devising hashing-based sampling algorithms for non-binary domains is an open problem. In

particular, sequence mining can be encoded with integer variables (Kemmar et al. 2014); generalized XOR constraints (Gomes et al. 2007a) is one possible research direction. Alternatively, as the M4RI library (Albrecht and Bard 2012) that we base our implementation on is optimized for *dense* \mathbb{F}_2 matrices, certain performance issues may be addressed with Gaussian elimination algorithms optimized for sparse matrices (Bouillaguet and Delaplace 2016).

Another limitation concerns the bounded tilt assumption regarding sampling distributions: many common quality measures, e.g., χ^2 , *information gain* (Nijssen et al. 2009), or *weighted relative accuracy* (Lemmerich et al. 2013), have high or even effectively infinite tilts (if φ can be arbitrarily close to 0). Such quality measures could be tackled with divide-and-conquer approaches (Chakraborty et al. 2014, Section 6) or alternative estimation techniques (Ermon et al. 2013b). This requires the capacity to efficiently handle constraints of the form $a \leq \varphi(p) \leq b$, which is possible for a number of quality measures, including the ones listed above.

9 Conclusion

We proposed FLEXICS, a flexible pattern sampler with theoretical guarantees regarding sampling accuracy. We leveraged the perspective on pattern mining as a constraint satisfaction problem and developed the first pattern sampling algorithm that builds upon the latest advances in sampling solutions in SAT. Experiments show that FLEXICS delivers the promised benefits regarding flexibility, efficiency, and sampling accuracy in itemset mining as well as in the novel task of pattern set sampling and that it is competitive with state-of-the-art alternatives.

Directions for future work include extensions to richer pattern languages and relaxing assumptions regarding sampling distributions (see Sect. 8 for a discussion). Specializing the sampling procedure towards typical mining scenarios may allow for deriving tighter theoretical bounds and improving the practical performance; examples include specific constraint types (e.g., anti-/monotone), shapes of sampling distributions (e.g., right-peaked distributions, similar to Fig. 5), and iterative mining. Following the future developments in weighted constrained sampling in AI may provide insights for improving various aspects of FLEXICS or pattern sampling in general.

Acknowledgements The authors would like to thank Guy Van den Broeck for useful discussions and Martin Albrecht for the support with the M4RI library. Vladimir Dzyuba is supported by FWO-Vlaanderen.

Appendix: WeightGen

In this section, we present an extended technical description of the WEIGHTGEN algorithm, which closely follows Sections 3 and 4 in Chakraborty et al. (2014), whereas the pseudocode in Algorithm 2 is structured similarly to that of UNIGEN2, a close cousin of WEIGHTGEN (Chakraborty et al. 2015). Lines 1–3 correspond to the estimation phase and Lines 4–8 correspond to the sampling phase. SOLVEBOUNDED stands for the bounded enumeration oracle.

Algorithm 2 WEIGHTGEN (Chakraborty et al. 2014)

Input: Boolean formula F , weight w , tilt bound \hat{r} , sampling error tolerance parameter κ
Assumes: $w(\cdot) \in [1/\hat{r}, 1]$, bounded enumeration algorithm SOLVEBOUNDED

```

1: for 17 iterations do                                     ▷ Corresponds to  $\delta_{est} = 0.2$ 
2:    $WeightEstimates \xleftarrow{Add} ESTIMATE(\emptyset)$ 
3:    $TotalWeight = MEDIAN(WeightEstimates)$ 
4:    $N_{XOR} = \mathcal{O}\left(\log_2 TotalWeight / (1 + \kappa^{-1})\right)$ 
5:    $loThresh \propto (1 + \kappa) / \kappa^2$ ,  $hiThresh \propto (1 + \kappa)^3 / \kappa^2$ 
6:   for  $N_{samples}$  times do
7:      $InitXORs = \{RANDOMXOR() \times N_{XOR} \text{ times}\}$ 
8:      $GENERATE(\kappa, [loThresh, hiThresh], InitXORs, 3)$ 

9: function  $ESTIMATE(XORs)$ 
  ▷ Returns an estimate of the total weight of all solutions
10:   $pivot_{est} = 46$                                           ▷ Corresponds to  $\varepsilon_{est} = 0.8$ 
11:   $Sols \leftarrow SOLVEBOUNDED(F, XORs, pivot_{est}, CellWeight \leftarrow \sum_{s \in Sols} w(s))$ 
12:  if  $CellWeight \leq pivot_{est}$  then                          ▷ Cell of the “right” size
13:    return  $CellWeight \times 2^{|XORs|}$ 
14:  else                                                       ▷ Shrink cell by adding XOR constraint
15:     $ESTIMATE(XORs \cup RANDOMXOR())$ 

16: function  $GENERATE(\kappa, [loThresh, hiThresh], XORs, i)$ 
  ▷ Returns a random solution of  $F$ 
17:   $Sols \leftarrow SOLVEBOUNDED(F, XORs, hiThresh), CellWeight \leftarrow \sum_{s \in Sols} w(s)$ 
18:  if  $CellWeight \in [loThresh, hiThresh]$  then                ▷ Cell of the “right” size
19:    return  $SAMPLEEXACTLY(Sols, w)$ 
20:  else if  $CellWeight > loThresh \wedge i > 0$  then                ▷ Cell is too large
21:     $GENERATE(\kappa, [loThresh, hiThresh], XORs \cup RANDOMXOR(), i - 1)$ 
22:  else                                                       ▷ Cell is too small
23:    return  $\perp$ 

```

The parameters of the estimation phase are fixed to particular theoretically motivated values. $pivot_{est}$ denotes the maximal weight of a cell at the estimation phase; $pivot_{est} = 46$ corresponds to estimation error tolerance $\varepsilon_{est} = 0.8$ (Line 10). If the total weight of solutions in a given cell exceeds $pivot_{est}$, a new random XOR constraint is added in order to eliminate a number of solutions. Repeating the process for a number of iterations increases the confidence of the estimate, e.g., 17 iterations result in $1 - \delta_{est} = 0.8$ (Line 1). Note that $ESTIMATE$ essentially estimates the total weight of *all* solutions, from which N_{XOR} , the initial number of XOR constraints for the sampling phase, is derived (Line 4).

A similar procedure is employed at the sampling phase. It starts with N_{XOR} constraints and adds at most *three* extra constraints. The user-chosen error tolerance parameter κ determines the range $[loThresh, hiThresh]$, within which the total weight of a suitable cell should lie (Line 5). For example, $\kappa = 0.9$ corresponds to range $[6.7, 49.4]$. If a suitable cell can be obtained, a solution is sampled exactly from all solutions in the cell; otherwise, no sample is returned. Requiring the total cell weight to exceed a particular value ensures the lower bound on the sampling accuracy.

The preceding presentation makes two simplifying assumptions: (1) all weights lie in $[1/r, 1]$; (2) adding XOR constraints never results in unsatisfiable subproblems

(empty cells). The former is relaxed by multiplying pivots by $\hat{w}_{max} = \hat{w}_{min} \times \hat{r} < 1$, where \hat{w}_{min} is the smallest weight observed so far. The latter is solved by simply restarting an iteration with a newly generated set of constraints. See Chakraborty et al. (2014) for the full explanation, including the precise formulae to compute all parameters.

Implementation details

Following suggestions of Chakraborty et al. (2015), we implement *leapfrogging*, a technique that improves the performance of the umbrella sampling procedure and thus benefits both GFLEXICS and EFLEXICS. First, after three iterations of the estimation phase, we initialize the following iterations with a number of XOR constraints that is equal to the smallest number returned in the previous iterations (rather than with zero XORs). Second, in the sampling phase, we start with one XOR constraint more than the number suggested by theory. If the cell is too small, we remove one constraint; if it is too large, we proceed adding (at most two) constraints. Both modifications are based on the observation that theoretical parameter values address hypothetical corner cases that rarely occur in practice. Finally, we only run the estimation phase until the initial number of XOR constraints, which only depends on the median of total weight estimates, converges. For example, if the estimation phase is supposed to run for 17 iterations, the convergence can happen as early as after 9 iterations.

References

- Aggarwal CC, Han J (eds) (2014) Frequent pattern mining. Springer International Publishing, New York
- Agrawal R, Mannila H, Srikant R, Toivonen H, Verkamo AI (1996) Fast discovery of association rules. In: Fayyad U, Piatetsky-Shapiro G, Smyth P, Uthurusamy R (eds) Advances in knowledge discovery and data mining. AAAI Press, Menlo Park, pp 307–328
- Albrecht M, Bard G (2012) The M4RI Library. The M4RI Team. <https://bitbucket.org/malb/m4ri>
- Berlingerio M, Pinelli F, Calabrese F (2013) ABACUS: frequent pattern mining-based community discovery in multidimensional networks. Data Min Knowl Discov 27(3):294–320
- Boley M, Grosskreutz H (2009) Approximating the number of frequent sets in dense data. Knowl Inf Syst 21(1):65–89
- Boley M, Gärtner T, Grosskreutz H (2010) Formal concept sampling for counting and threshold-free local pattern mining. In: Proceedings of the 10th SIAM international conference on data mining (SDM '10), pp 177–188
- Boley M, Lucchese C, Paurat D, Gärtner T (2011) Direct local pattern sampling by efficient two-step random procedures. In: Proceedings of the 17th ACM SIGKDD conference on knowledge discovery and data mining (KDD '11), pp 582–590
- Boley M, Moens S, Gärtner T (2012) Linear space direct pattern sampling using coupling from the past. In: Proceedings of the 18th ACM SIGKDD conference on knowledge discovery and data mining (KDD '12), pp 69–77
- Boley M, Mampaey M, Kang B, Tokmakov P, Wrobel S (2013) One click mining—interactive local pattern discovery through implicit preference and performance learning. In: Proceedings of the ACM SIGKDD workshop on interactive data exploration and analytics (IDEA '13), pp 28–36
- Bonchi F, Giannotti F, Lucchese C, Orlando S, Perego R, Trasarti R (2009) A constraint-based querying system for exploratory pattern discovery. Inf Syst 34(1):3–27
- Bouillaguet C, Delaplace C (2016) Sparse Gaussian elimination modulo p : an update. In: Proceedings of the 18th international workshop on computer algebra in scientific computing (CASC '16), pp 101–116

- Bringmann B, Nijssen S, Tatti N, Vreeken J, Zimmermann A (2010) Mining sets of patterns. In: Tutorial at the European conference on machine learning and principles and practice of knowledge discovery (ECML/PKDD '10)
- Bucilă C, Gehrke J, Kifer D, White W (2003) Dualminer: a dual-pruning algorithm for itemsets with constraints. *Data Min Knowl Discov* 7(3):241–272
- Calders T, Rigotti C, Boulicaut JF (2006) A survey on condensed representations for frequent sets. In: Boulicaut JF, De Raedt L, Mannila H (eds) *Constraint-based mining and inductive databases*. Springer, Berlin, pp 64–80
- Carvalho DR, Freitas AA, Ebecken N (2005) Evaluating the correlation between objective rule interestingness measures and real human interest. In: *Proceedings of the 9th European conference on principles of data mining and knowledge discovery (PKDD '05)*, pp 453–461
- Chakraborty S, Meel KS, Vardi MY (2013) A scalable and nearly uniform generator of SAT witnesses. In: *Proceedings of the 25th international conference on computer-aided verification (CAV '13)*, pp 608–623
- Chakraborty S, Fremont DJ, Meel KS, Vardi MY (2014) Distribution-aware sampling and weighted model counting for SAT. In: *Proceedings of the 28th AAAI conference on artificial intelligence (AAAI '14)*, pp 1722–1730
- Chakraborty S, Fremont DJ, Meel KS, Seshia SA, Vardi MY (2015) On parallel scalable uniform SAT witness generation. In: *Proceedings of the 21st international conference on tools and algorithms for the construction and analysis of systems (TACAS '15)*, vol 9035, pp 304–319
- De Raedt L, Zimmermann A (2007) Constraint-based pattern set mining. In: *Proceedings of the 7th SIAM international conference on data mining (SDM '07)*, pp 237–248
- Dzyuba V, van Leeuwen M (2017) Learning what matters—sampling interesting patterns. In: *Proceedings of the 21st Pacific-Asia conference on knowledge discovery and data mining (PAKDD '17)* (**in press**)
- Ermon S, Gomes CP, Sabharwal A, Selman B (2013a) Embed and project: discrete sampling with universal hashing. *Adv Neural Inf Process Syst* 26:2085–2093
- Ermon S, Gomes CP, Sabharwal A, Selman B (2013b) Taming the curse of dimensionality: discrete integration by hashing and optimization. In: *Proceedings of the 30th international conference on machine learning (ICML '13)*, pp 334–342
- Geerts F, Goethals B, Mielikäinen T (2004) Tiling databases. In: *Proceedings of the 7th international conference on discovery science (DS '04)*, pp 278–289
- Giacometti A, Soulet A (2016) Anytime algorithm for frequent pattern outlier detection. *Int J Data Sci Anal* 2(3):119–130
- Gomes CP, van Hoeve Wj, Sabharwal A, Selman B (2007a) Counting CSP solutions using generalized XOR constraints. In: *Proceedings of the 22nd AAAI conference on artificial intelligence (AAAI '07)*, pp 204–209
- Gomes CP, Sabharwal A, Selman B (2007b) Near-uniform sampling of combinatorial spaces using XOR constraints. *Adv Neural Inf Process Syst* 19:481–488
- Guns T, Nijssen S, De Raedt L (2011) Itemset mining: a constraint programming perspective. *Artif Intell* 175(12–13):1951–1983
- Guns T, Nijssen S, De Raedt L (2013) *k*-Pattern set mining under constraints. *IEEE Trans Knowl Data Eng* 25(2):402–418
- Hasan MA, Zaki MJ (2009) Output space sampling for graph patterns. *Proc VLDB Endow* 2(1):730–741
- Kemmar A, Ugarte W, Loudni S, Charnois T, Lebbah Y, Boizumault P, Crémilleux B (2014) Mining relevant sequence patterns with CP-based framework. In: *Proceedings of the 26th IEEE international conference on tools with artificial intelligence (ICTAI '14)*, pp 552–559
- Khiari M, Boizumault P, Crémilleux B (2010) Constraint programming for mining *n*-ary patterns. In: *Proceedings of the 16th international conference on principles and practice of constraint programming (CP '10)*, pp 552–567
- Knobbe A, Ho E (2006) Pattern teams. In: *Proceedings of the 10th European conference on principles of data mining and knowledge discovery (PKDD '06)*, pp 577–584
- Lemmerich F, Becker M, Puppe F (2013) Difference-based estimates for generalization-aware subgroup discovery. In: *Proceedings of the European conference on machine learning and principles and practice of knowledge discovery (ECML/PKDD '13)*, pp 288–303
- Meel K, Vardi M, Chakraborty S, Fremont D, Seshia S, Fried D, Ivrii A, Malik S (2016) Constrained sampling and counting: universal hashing meets SAT solving. In: *Proceedings of the beyond NP AAAI workshop*

- Nijssen S, Zimmermann A (2014) Constraint-based pattern mining. In: Aggarwal CC, Han J (eds) *Frequent pattern mining*, chap 7. Springer International Publishing, New York, pp 147–163
- Nijssen S, Guns T, De Raedt L (2009) Correlated itemset mining in ROC space: a constraint programming approach. In: *Proceedings of the 15th ACM SIGKDD conference on knowledge discovery and data mining (KDD '09)*, pp 647–655
- Paramonov S, van Leeuwen M, Denecker M, De Raedt L (2015) An exercise in declarative modeling for relational query mining. In: *Proceedings of the 25th international conference on inductive logic programming (ILP '15)*
- Pei J, Han J (2000) Can we push more constraints into frequent pattern mining? In: *Proceedings of the 6th ACM SIGKDD conference on knowledge discovery and data mining (KDD '00)*, pp 350–354
- Ramakrishnan N, Kumar D, Mishra B, Potts M, Helm R (2004) Turning CARTwheels: an alternating algorithm for mining redescrptions. In: *Proceedings of the 10th ACM SIGKDD conference on knowledge discovery and data mining (KDD '04)*, pp 266–275
- Shervashidze N, Vishwanathan S, Petri T, Mehlhorn K, Borgwardt KM (2009) Efficient graphlet kernels for large graph comparison. In: *Proceedings of the 12th international conference on artificial intelligence and statistics (AISTATS '09)*, pp 488–495
- Soos M (2010) Enhanced Gaussian elimination in DPLL-based SAT solvers. In: *Proceedings of the pragmatics of SAT workshop (POS '10)*, pp 2–14
- Uno T, Kiyomi M, Arimura H (2005) LCM ver. 3: collaboration of array, bitmap and prefix tree for frequent itemset mining. In: *Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations (OSDM '05)*, pp 77–86
- Zaki MJ, Parthasarathy S, Ogihara M, Li W (1997) New algorithms for fast discovery of association rules. In: *Proceedings of the 3rd ACM SIGKDD conference on knowledge discovery and data mining (KDD '97)*, pp 283–296
- Zimmermann A, Nijssen S (2014) Supervised pattern mining and applications to classification. In: Aggarwal CC, Han J (eds) *Frequent pattern mining*, chap 17. Springer International Publishing, New York, pp 425–442