Accelerating Itemset Sampling using Satisfiability Constraints on FPGA

Mael Gueguen, Olivier Sentieys, Alexandre Termier

Accelerating Itemset Sampling using Satisfiability Constraints on FPGA

Mael Gueguen*, Olivier Sentieys and Alexandre Termier
Univ Rennes, Inria, CNRS, IRISA,
Rennes, France
Email: *mael.gueguen@irisa.fr

***Abstract*—Finding recurrent patterns within a data stream is important for fields as diverse as cybersecurity or e-commerce. This requires to use pattern mining techniques. However, pattern mining suffers from two issues. The first one, known as "pattern explosion", comes from the large combinatorial space explored and is the result of too many patterns outputed to be analyzed. Recent techniques called output space sampling solve this problem by outputing only a sampled set of all the results, with a target size provided by the user. The second issue is that most algorithms are designed to operate on static datasets or low throughput streams. In this paper, we propose a contribution to tackle both issues, by designing an FPGA accelerator for pattern mining with output space sampling. We show that our accelerator can outperform a state-of-the-art implementation on a server class CPU using a modest FPGA product.**

I. Introduction

When analyzing data, an important task is to discover *correlations*. Such correlations can be that Amazon customers that buy a laptop often buy a mouse simultaneously, or that a set of files are often accessed simultaneously in a cloud storage context. These correlations give powerful insights on the contents of the data, that may be used to take actions. For example Amazon will immediately show mouses to customers having put a laptop in their cart, and the cloud storage operator will position the files often accessed together on the same physical machine.

Discovering potentially complex correlations is handled by Pattern Mining algorithms, whose goal is to explore a huge combinatorial space efficiently. Most of them have been designed for an offline data analysis setting: a static dataset is input to the algorithm which performs pattern extraction (runtime varying from a few seconds to several days depending on the data and parameters). The results are then manually reviewed by a data analyst. However, many modern settings require an online loop, where the data is continuously arriving. In this context, patterns are extracted in few seconds at most, and are used to take automated decisions. For example, Amazon needs to be able to adapt quickly to changes in customer behavior or arrival of new products, and file access patterns will change upon completion of projects using those files.

There exist some pattern mining algorithms that handle data streams, however they have two limitations: 1) they are likely to output thousands of results of mixed interest and 2) they are not designed to cope with a high throughput. The first limitation is shared by most pattern mining algorithms: they extract patterns that are repeated *frequently* in the data, and in practice there are many of such repeated patterns. A large portion of these patterns contain redundant information. There are several ways to reduce the number of patterns output to a manageable size, one of the most drastic being *pattern sampling*. The idea is to only compute a set of frequent patterns, and to have statistical guarantees that these patterns are a "good" sample of the complete set of patterns. Pattern sampling algorithms are a promising approach for performing a pattern-based analysis of data streams.

Our contribution is to show that pattern sampling can be performed extremely efficiently on an FPGA. This paves the way for future pattern sampling hardware accelerators, able to extract on the fly patterns from data streams with a very high throughput. More precisely, we adapted the state-of-the-art algorithm Flexics [1] for execution on an FPGA. We explain how to sample randomly itemsets while keeping a control flow applicable for FPGA logic.

The paper is organized as follows. Section II introduces the pattern mining problem, the Eclat algorithm, and pattern sampling. Section III describes the state of the art Flexics algorithm for pattern sampling, as well as our algorithmic contribution to adapt Flexics to FPGA. Section IV presents the architecture of our approach. Results are presented in Section V. Last, Section VI concludes the paper and hints at future works.

II. Background and related work

We first introduce the general pattern mining problem, as well as the Eclat algorithm for mining frequent itemsets. We then present existing approaches for accelerating pattern mining algorithms on FPGA and expose pattern sampling approaches.

A. Pattern mining

Consider an alphabet $\mathcal{A} = \{X_i\}$, with $i \in [\![1, N]\!]$. The elements of $\mathcal{A}$ are called *items*, and any subset $I \subseteq \mathcal{A}$ is called an *itemset*. A database $\mathcal{D}$ is a multiset of *transactions*, each transaction being an itemset of $\mathcal{A}$. The *support* of an itemset $I \subseteq \mathcal{A}$ in $\mathcal{D}$ corresponds to the number of transactions of $\mathcal{D}$ containing I. An itemset is *frequent* in $\mathcal{D}$ if its support is above a user-defined frequency threshold t. There is a vast literature on algorithms for mining frequent itemsets. In

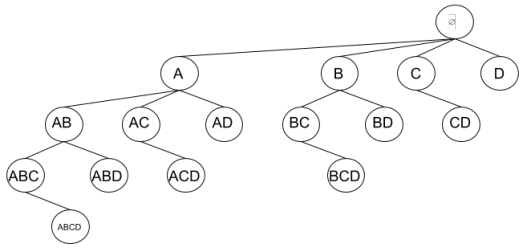

Fig. 1. Example of a depth-first tree for the alphabet $\{a, b, c, d\}$ in ECLAT

this paper, we focus on the Eclat algorithm [2]. Like most pattern mining algorithms, Eclat follows a "generate and test" approach: potentially frequent itemsets are generated, then their frequency is tested in the database. Eclat explores the search space of candidate itemsets in a depth-first manner, starting with small itemsets and augmenting them one item at a time. An example of a search tree is given in Figure 1. Note that the tree is asymmetric to avoid exploring twice the same itemset.

ECLAT relies on the *Apriori* principle [3] to prune the search space. If an itemset is infrequent in the database, any itemset that contains it will have a lower frequency and it will be infrequent too.

With the search space of Figure 1, consider the database $\mathcal{D} = \{abc, bd, abd, acd\}$ and $t = 0.5$. The itemset ab is present twice, thus has a frequency of $0.5 \geq t$, and is frequent: its children in the tree will be tested. On the other hand, the itemset abc is present only once, thus has a frequency of $0.25 < t$ and is infrequent. Its children in the tree do not need to be generated nor tested.

B. Itemset mining on FPGA

Two families of methods are described in the literature for itemset mining on FPGA. A first method is to use a controller like a CPU allowing for complex controls, and restrict the FPGA to accelerate the support counting of the itemsets [4]. The other methods accelerate the whole application on hardware [5], [6]. The main difficulty is to streamline the exploration of the tree-shaped search space for an efficient execution on FPGA. The asymetry of the tree is a first problem. The second one is that pruning by the Apriori property, while strictly mandatory for performance, makes the search tree unpredictably irregular. The approach proposed by [6] elegantly solves both problems, and we used it as a starting point in the design of our approach for FPGA-based pattern sampling.

C. Pattern Sampling

Output space sampling is tailored to return a bounded number of patterns without altering the database. Most approaches provide statistical guarantees that the distribution of the sampled patterns is similar to the distribution of the complete set of patterns. Earlier works have adapted standard statistical sampling methods to the case of frequent itemsets [7], or exploit simple statistical observations on the structure of the pattern space [8].

More recently, Dzyuba *et al.* [1] proposed the Flexics framework. Inspired by sampling strategies used in SAT (Boolean satisfiability) solvers, this approach is built on top of standard (non-sampling) frequent itemset mining algorithms. Flexics adds random XOR constraints that exclude some frequent itemsets from the result. These constraints have the property to partition the frequent itemset space in "cells", where the distribution of itemsets inside each cell is similar to the distribution of the complete space. Given an expected number of frequent itemsets in the output, Flexics will add as many random XOR constraints as necessary to get to a cell containing this number of frequent itemsets (within a small margin of approximation).

The Flexics framework is one of the most efficient for principled pattern sampling. Furthermore, the original paper shows how to build it above Eclat, an FPGA-friendly algorithm (compared to other frequent itemset miners). And the approach is based on XOR constraints, which are well handled by hardware. It thus seems especially relevant to base an FPGA-based pattern sampling method on the Flexics framework. To the best of our knowledge, this is the first pattern sampling accelerator on FPGA architectures. This is our contribution in this paper, presented in the next sections.

III. Flexics algorithm for FPGA

This section explains our algorithmic contribution to adapt Flexics for FPGA, while the hardware architecture of our solution will be explained in the following section. We first explain how constraints are generated by Flexics. The two step approach proposed in software cannot be used on FPGA, we thus present our first improvement that allows dynamic generation of constraints on the FPGA. We then show how a major improvement of the constraint system can also be used with our approach. Finally, we explain how the constraints can be exploited in an FPGA version of Eclat.

A. Sampling Itemsets with a System of Binary Constraints

As introduced in the previous section, the main idea of Flexics is to use random XOR constraints to cut the itemset space in regions of equivalent size and distribution. The original Flexics algorithm is a two step process, where first the *Weightgen oracle* is called, followed by the actual itemset mining step. The Weightgen oracle, described in [9], estimates the *weight* of the solution (related to the number of results found) of a SAT problem. Weightgen can also be used to generate random arbitrary constraints on the variables of the SAT problem that will reduce the number of results, until the estimated weight is lower than or equal to a user specified threshold. In the case of Flexics, the SAT problem given to Weightgen is finding frequent itemsets in the database. The random constraints generated to sample the results are XOR constraints using the alphabet $\mathcal{A}$ of size N from the database. Generated XOR constraints have the form

$$\bigoplus_{i \in [\![1,N]\!]} b_i.X_i = b_0, \tag{1}$$

where b_i are randomly generated independent variables in $[\![0,1]\!]$ with a uniform distribution. For any $i \in [\![1,N]\!]$, a b_i dictates if the item X_i will be present in the XOR constraint (1 if present, 0 otherwise). The b_0 coefficient is called the *parity bit*, as a XOR with multiple inputs returns the parity of the sum of its boolean inputs. These XOR constraints acts as a system of boolean equations, where a given pattern can either satisfy all the equations for the system, or not satisfy at least one of them.

For example, with a database using an alphabet $\mathcal{A}' = \{a, b, c, d, e, f\}$, and given two randomly generated XOR constraints $a \oplus c \oplus e = 1$ and $b \oplus e \oplus f = 0$, we can check if two frequent candidate itemsets $abde$ and $abdf$ are to be outputed in the samples. For $abde$, the first XOR constraint gives $(1 \oplus 0) \oplus 1 = 1 \iff (1) \oplus 1 = 1 \iff 0 = 1$, which is false. Thus, $abde$ does not satisfy all XOR constraints and is not part of the subspace to be be outputed. For $abdf$, the first XOR constraint gives $(1 \oplus 0) \oplus 0 = 1 \iff (1) \oplus 0 = 1$, which is true, and the second one gives $(1 \oplus 0) \oplus 1 = 0 \iff (1) \oplus 1 = 0 \iff 0 = 0$, which is true. Thus, the frequent itemset $abdf$ satisfies all XOR constraints and will be outputed as a sample.

The two step approach (Weightgen then mining) is not relevant on an FPGA. First, implementing Weightgen would require a lot of estate, that would be unused most of the time. Second, in a streaming context one would like to be able to quickly react to changes in the pattern distribution, which is difficult to do when pre-computing the number of constraints. Our first contribution is thus to propose a single step approach for pattern sampling, that does not use Weightgen. Instead, the mining starts immediately, and during the mining, our approach dynamically generates new XOR constraints if the potential output size goes above the user-given threshold.

This way, the number of interesting patterns does not need to be estimated in advance, but the list of interesting patterns discovered has to be maintained when new constraints are generated by dynamically pruning patterns than do not satisfy the newly added constraints. The principles of our dynamic constraint generation is as follows. First, new XOR constraints are generated randomly. Our algorithm ensures that no constraint in the system of equations can be derived from a linear combination of the other constraints. If a newly generated XOR constraint causes the system of equations to be unsolvable, it is rejected. If it is equal to a linear combination of already generated constraints, the new constraint is also rejected. After a constraint is rejected, new constraints are randomly generated, until a generated random XOR constraint respects the criteria. In practice, however, there is a probability of $\frac{1}{2^{N-K}}$ for a new random XOR constraint to be rejected in a system already holding K XOR constraints. Thus, this process of rejecting constraints does not impact significantly average performance.

B. Rewriting the Constraints with Gauss-Jordan Elimination

The problem we try to solve is the following: scan the depth-first tree with the ECLAT algorithm, but skip the itemsets that do not satisfy the generated XOR constraints. However, when scanning the sparse samples in the tree, there is no simple relation like the prefix/suffix separation used in [6] between a sample and the next.

It is harder to compute what is the next sample to test given any position in the tree, and subsequently, to know if there exist satisfying itemsets down its branch or if the branch is finished. We developed a solution to reduce the problem of mining satisfying itemsets randomly distributed in the tree to a simpler yet equivalent problem that uses ECLAT on a reduced tree that represents *at least* all frequent satisfying itemsets.

At any given time with K generated XOR constraints, for an alphabet of size N, the corresponding system of equations would be in the form of

$$\forall k \in [\![1,K]\!],\ \left(\bigoplus_{i \in [\![1,N]\!]} b_{i,k}.X_i \right) = b_{0,k}, \tag{2}$$

where $b_{i,k} \in [\![0,1]\!]$, $\forall k \in [\![1,K]\!]$ and $\forall i \in [\![1,N]\!]$. With this notation, the $b_{i,k}$ can be considered as matrix coefficients for an array of size $K \times (N+1)$. The parameter K is dynamic and data dependant (it increases as new samples are found) and is only theoretically bounded by N.

Using linear combinations, the matrix corresponding to the system of equations can be transformed using Gauss-Jordan elimination, to generate a row echelon matrix. The solutions of the original system of equations and after the Gauss-Jordan elimination are strictly the same. The Gauss-Jordan elimination is a variant of Gaussian elimination which results in a unique normalized row echelon matrix. This unique matrix has two types of rows: one type can contain any value, while the other type of row contains exactly one 1 that delimits the echelon and 0 everywhere else. After a Gauss-Jordan elimination, our algorithm rewrites the system of equations in the form of

$$\forall k \in [\![1,K]\!],\ X_{c_k} = \left(\bigoplus_{i \in [\![1,N-K]\!], f_i > c_k} b'_{f_i,k}.X_{f_i} \right) \oplus b'_{0,k} \tag{3}$$

where $\{c_i\}_{i \in [\![1,K]\!]}$ denote the indexes of the rows corresponding to the echelons. There are K indexes c_i as there are K rows in the Gauss-Jordan matrix. We refer to the items X_{c_i} as *constrained items*. All the other indexes corresponding to rows with any values allowed are noted as $\{f_i\}_{i \in [\![1,N-K]\!]}$. There are $N-K$ indexes f_i as there are N indexes in total. The items with indexes X_{f_i} are called *free items*.

C. Scanning sample space with a bijective transformation

For any system of equations resulting from XOR constraints, our approach is to cut the alphabet into two parts:

- the set of constrained items X_{c_k} that appear in only one XOR constraint after Gauss Jordan elimination and
- the set containing the rest of the alphabet, denoted free alphabet, whose items X_{f_i} can appear in any XOR constraint.

In this way, there are two complementary alphabets, that can be non-contiguous, splitting items from the initial alphabet in two categories. Then, the right hand of Equation 3 is computed using the $b'_{f_i,k}$ in the Gauss-Jordan Matrix and the X_{f_i} are equal to 1 if the f_i are present in the free itemset and to 0 otherwise. This formula dictates if the items X_{c_k} are present in the constrained itemset or not.

For two different itemsets constructed from $\mathcal{A}$ that satisfy the XOR constraints, their free itemsets are necessarily different. Reciprocally, there exists a single itemset constructed from $\mathcal{A}$ that satisfies the XOR constraints for each free itemset. Thus, there is a bijection between the pattern space of the database alphabet and the subspace of patterns satisfying the XOR constraints: a scan of all free itemsets can be used to scan all satisfying itemsets. Furthermore, a sample constructed from the full alphabet $\mathcal{A}$ will always be a super-set of its reduced free itemset. Thus, the Apriori principle dictates that if a sample itemset is frequent, its corresponding free itemset is also frequent. Thus, a scan of all *frequent* free itemsets can be used to scan all *frequent* satisfying itemsets. Frequent free itemsets can lead to infrequent satisfying itemsets, but the algorithm is guaranteed to find all samples.

Finding all frequent free itemsets can be done using ECLAT on the reduced free alphabet, and for each frequent free itemset, the algorithm will test its satisfying counterpart. For each itemset in the "reduced" depth-first tree, the accelerator tests the frequency of two itemsets. The free itemset has to be tested to allow pruning with the Apriori principle, and the sample itemsets have to be tested since they will be returned as the final result.

IV. Accelerator Architecture

This section presents the architecture of our accelerator on an actual FPGA board, and gives details on how the solution is implemented.

A. Overview

Figure 2 shows an overview of the accelerator architecture implemented in the programmable logic fabric of a Zynq FPGA from Xilinx. The *zc702* evaluation board used for the experiments holds a *System on Chip*, containing two Cortex A-9 CPUs and a XC7Z020-CLG484-1 FPGA, connected to an off-chip 1GB DDR3 memory. Our design uses communications between the FPGA and the CPUs only when the hardware application starts and stops, in order to interface with the user. All accesses from the accelerator in the FPGA to the main memory are achieved using the *High Performance AXI* port, without cache coherency protocols. This section details the support counting acceleration of Figure 2 and an additional feature we called correspondence list. The candidate generation mechanism is based on [6], improved to support XOR constraints, and will not be detailed here.

B. Support Counting

The support counting loop of the architecture takes most of the execution time, given the finite memory bandwidth. The

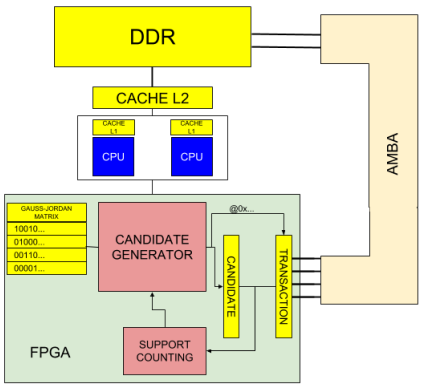

Fig. 2. Overview of the accelerator architecture

loop takes in transactions from the database, checks if the current candidate is present, and if so, increments a counter. Given a transaction bitvector and a candidate itemset bitvector, for the candidate to be present in the transaction, all bits set to 1 in the candidate have to be set to 1 in the transaction too. If we call the current candidate itemset to test c, and the current transaction from the database t, one way to implement this proposition with FPGA logic is

$$c \subset t \iff t\& \; c = c. \tag{4}$$

This proposition can be simplified, since all bits set to 1 are set to 0 in $\bar{t}$. Thus for all items in c, they have to be absent from $\bar{t}$ for c to be present in t:

$$c \subset t \iff \bar{t} \; \& \; c = 0. \tag{5}$$

Since this feature only uses simple logic gates and a counter increment, its impact on the area cost is very low.

C. Correspondence List Address Cache

Many dataset reduction techniques have been developed for pattern mining in order to exclude redundant or unnecessary operations. Several of them can be seen as preprocessing techniques to optimize the ensuing pattern mining, such as removing non-frequent items from the database, or grouping identical transactions together. Some others aim to optimize the mining of patterns using knowledge discovered during the pattern mining.

In the special case of a depth-first algorithm like ECLAT, if the set of transactions containing a frequent itemset is registered in the memory of the FPGA, it is guaranteed the next candidate itemset is a child of the itemset (when it is not a leaf of the tree), and its own set of transactions is contained in the registered set of transactions. By allocating in memory the addresses of transaction containing an itemset, the accelerator is able to scan a reduced portion of the database, as illustrated in Figure 3. If the set of addresses cannot fit in the allocated memory, the accelerator scans the whole database.

This technique can only work if the dataset uses a frequency threshold sufficiently low to fit in the FPGA memory. If the accelerator is tasked to find itemsets present in at least 90% of a large database, the correspondence list would not fit in the

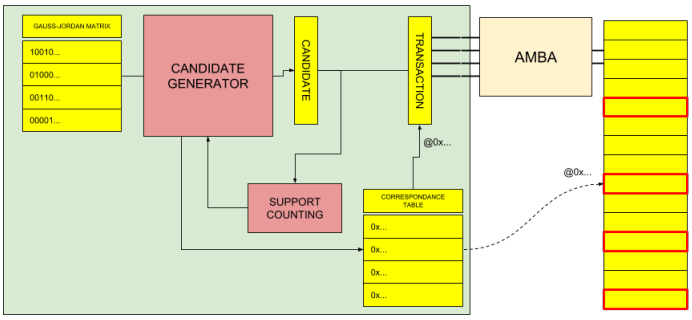

Fig. 3. Example of dataset reduction using correspondence list

FPGA. The pattern explosion implies that a regular itemset miner could spend the majority of its execution scanning for itemsets with very low support (sometimes less than thousands or hundred of transactions depending on the database). Our proposition is to speedup the computation of these itemsets with low support, but with support still superior to the frequency threshold, when the density of the dataset favors it.

If we allow to register only the set of transactions of several frequent itemsets, it is possible to store the histories of frequent itemsets at different depths in the tree. this reduces the number of transactions accessed in memory. In order to only keep track of set of transactions of itemsets that can be reused by future candidate itemsets, the addresses of transactions of an itemset is evicted from memory depending on its depth in the tree.

D. Parameters' Influence on the Accelerator

In order to analyze the theoretical performance of the accelerator, we introduce two parameters: DB_SIZE, the number of transactions in the database, and NB_BLOCK, the number of memory accesses to get a transaction. During the support counting, the accelerator processes chunks of $\frac{N}{NB\_BLOCK}$ bits wide data in a pipelined fashion. If the memory is able to feed the accelerator without bandwidth limitation, the pipeline can process a chunk every four cycles at 100MHz. The number of cycles needed to compute the support of an itemset is $4 \times DB\_SIZE \times NB\_BLOCK$.

If we consider an example where the memory has to deliver chunks of size $N/NB\_BLOCK = 128$ bits, the internal logic of the FPGA can process $128/4 \times 100Mb/s = 400MB/s$. Thus, the accelerator will be stalled by the memory if it is unable to reach this throughput. This is often the case in I/O bound applications such as data mining. A simple way to tackle this issue is to encode the transactions inside the memory, and decode it back to bit vectors once in the FPGA. The naive approach to store potentially long bit vectors can be a huge waste of space and bandwidth, especially with sparse datasets containing many zeroes.

Figure 4 shows the amount of resources allocated on the XC7Z020 SoC for different values of N. An accelerator with a fixed value N can process any database with an alphabet of size less or equal to N. At the moment only rather small alphabets can be processed because of the high usage of Look-Up Tables (LUT), as nearly all of the slices are allocated for an

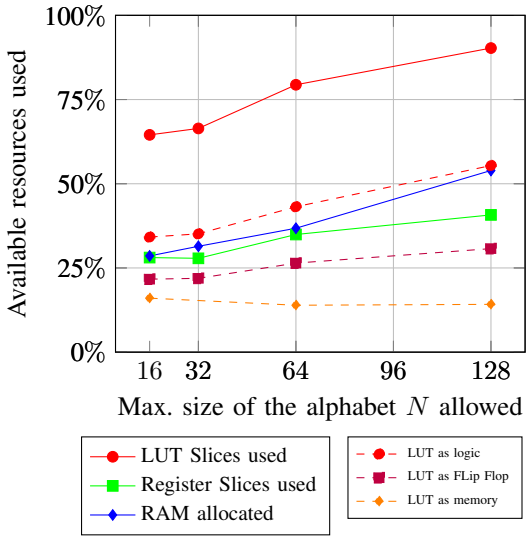

Fig. 4. Execution time wrt. correspondence list size

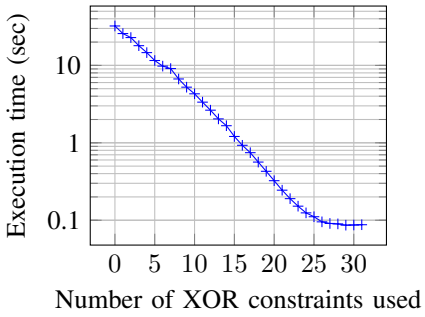

Fig. 5. Impact of number of XOR constraints on execution time

alphabet of size 128. For the same alphabet size, only half of the available RAM and register slices are used. The parameter N is thus linked to two limiting factors: the allocated logic and the memory bandwidth. It is however noteworthy that the FPGA used in our experiments is rather small and our approach will scale efficiently with larger FPGA devices.

V. Experiments

The dataset used for the performance analysis of our accelerator is available publicly from [10]. This dataset uses an alphabet of size 32. We impose $N/NB\_BLOCK = 32$ bits. The FPGA thus processes $100MB/s$, which is close to the bandwidth between the memory and the FPGA. The dataset consists of around 13000 transactions, and the frequency threshold used is 1%. This fits very well the use case of a low-frequency support, and allows for our design to be tested with a wide range of correspondence list sizes.

A. Impact of Sampling on Execution Time

Figure 5 shows the effects of the sampling on the execution time. In this scenario, the accelerator can return any number of samples (no user bound), but the number of random XOR constraints is fixed at compilation. Each new constraint

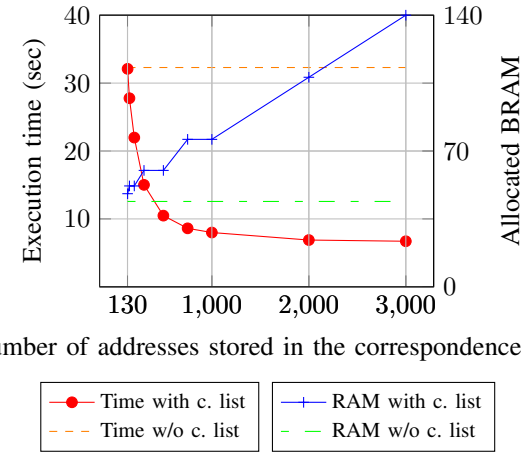

Fig. 6. Execution time wrt. correspondence list size

reduces the execution time by a factor close to 1.2, until it reaches the point where there are so few itemsets left to scan, the initialization becomes non-negligible.

B. Impact of Correspondence List Size

Figure 6 compares the execution time of the accelerator with and without a correspondence list. The correspondence list brings a very noticeable speedup with rather low amounts of memory allocated. Allocating memory to store 300 addresses (roughly twice the support threshold in this case) results in a speedup of 2.1. The maximum speedup corresponding to 3000 addresses reaches 4.7. Figure 6 also displays the amount of memory allocated in the FPGA for the tested values. The relation is not stricly linear because only fixed amounts of block RAM can be used. The correspondence list with a size of 3000 results in all the BRAM (140) being allocated.

TABLE I
TIME NEEDED (IN SECONDS) TO COMPUTE 1000 SAMPLES

	EFlexics (CPU)	Our FPGA
german	25	11,7
heart	45	11,6
kr-vs-kp	9	12,7
primary	10	1,2
vote	19	5,8

C. Comparison Against Flexics

Table I shows results taken from [1], where EFlexics (the fastest implementation of Flexics) is tasked to return 1000 samples. These results were obtained on an Intel Xeon CPU running at 3.2GHz and with 32Gb of RAM on the CP4IM dataset [11]. We compare the execution of our accelerator running on the *zc702* board when tasked to return a thousand samples to the software execution time, without allocating a correspondence table. Our accelerator achieves a speedup between 2 and 10, except for the *kr-vs-kp* dataset. The accelerator performs best with small alphabet sizes, as these problems require a lot less memory bandwidth.

VI. CONCLUSION

We proposed a first approach for performing pattern sampling on FPGA. This approach is promising: despite running on a modest FPGA clocked at 100 MHz, it can be up to an order of magnitude faster than a software version running on a server class Xeon CPU clocked at 3.2 GHz. We also proposed a dataset reduction technique applicable to depth-first algorithms as ECLAT to improve execution time, at the cost of allocated memory on the FPGA.

The current approach is sequential: an exciting research direction is to parallelize the implementation in order to explore several branches of the search space simultaneously. Since itemsets access to the same database, the transactions can be used for the support counting of multiple itemsets. These two properties allow to divide the whole tree in subtrees that can be processed in parallel without requiring more communication between the FPGA and the memory.

This parallel extension of the proposed accelerator will allow to make good use of higher-end FPGA and reach throughputs compatible with real-time analysis on demanding streaming data. The current paper focused on the FPGA algorithm and its performance. In order to process actual streams, another important element is to have efficient communications between the RAM holding the data and the FPGA. In this regard, a solution is to compress the database in RAM, which will cost a small allocation of resources on the FPGA to decode the transactions, but will allow a much better use of the bus bandwidth.

REFERENCES

[1] V. Dzyuba, M. van Leeuwen, and L. De Raedt, "Flexible constrained sampling with guarantees for pattern mining," *Data Mining and Knowledge Discovery*, Mar. 2017.
[2] M. J. Zaki *et al.*, "New algorithms for fast discovery of association rules." in *Proc. ACM Int. Conf. on Knowledge Discovery and Data Mining (KDD)*, vol. 97, 1997, pp. 283–286.
[3] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *Proceedings of 20th International Conference on Very Large Data Bases (VLDB)*, 1994, pp. 487–499.
[4] A. Prost-Boucle, F. Pétrot, V. Leroy, and H. Alemdar, "Efficient and versatile fpga acceleration of support counting for stream mining of sequences and frequent itemsets," *ACM Trans. on Reconfigurable Technol. and Syst. (TRETS)*, vol. 10, no. 3, p. 21, 2017.
[5] S. Shi, Y. Qi, and Q. Wang, "Accelerating intersection computation in frequent itemset mining with fpga," in *IEEE 10th Int. Conf. on High Perf. Comp. and Comm. (HPCC)*, 2013, pp. 659–665.
[6] Y. Zhang, F. Zhang, Z. Jin, and J. D. Bakos, "An fpga-based accelerator for frequent itemset mining," *ACM Trans. Reconfigurable Technol. Syst. (TRETS)*, vol. 6, no. 1, pp. 2:1–2:17, May 2013.
[7] M. Al Hasan and M. J. Zaki, "Output Space Sampling for Graph Patterns," *Proc. VLDB Endow.*, vol. 2, no. 1, pp. 730–741, Aug. 2009.
[8] M. Boley, S. Moens, and T. Gärtner, "Linear Space Direct Pattern Sampling Using Coupling from the Past," in *Proc. 18th ACM Int. Conf. on Knowledge Discovery and Data Mining (KDD)*, 2012, pp. 69–77.
[9] S. Chakraborty, D. J. Fremont, K. S. Meel, S. A. Seshia, and M. Y. Vardi, "Distribution-Aware Sampling and Weighted Model Counting for SAT," *arXiv:1404.2984*, Apr. 2014.
[10] D. Dheeru and E. Karra Taniskidou, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml
[11] T. Guns, S. Nijssen, and L. De Raedt, "Itemset mining: A constraint programming perspective," *Artificial Intelligence*, vol. 175, no. 12-13, pp. 1951–1983, 2011.