Discovering Periodicity in Locally Repeating Patterns

Abstract—**Analysing and learning from sequentially ordered symbolic data is increasingly important in applications such as finance and biology. Finding patterns that show interesting behaviour, such as regularly repeating occurrences within a time interval, can provide useful insight. In this paper we address the problem of efficiently identifying such behaviours. Existing approaches often require a target period to be specified, which will limit possible patterns to those approximating the specified periodicity, such as daily, monthly, quarterly and so on. In this paper we extend one such approach, derived from frequent pattern mining, to operate without the need for user-specified periodicity. Our new algorithms can identify the time interval and periodicity, or frequency of occurrence, of all periodically occurring patterns within a certain used-specified tolerance. Experimental results of our implementation show that the new approach can identify many more patterns in a real-world financial dataset, while on other sequential datasets it finds similar numbers of patterns without significant reduction in efficiency compared to existing approaches. We also verify the algorithm's ability to recover recurring patterns in controlled experiments on synthetic data.**

Index Terms—**sequential pattern mining, periodic patterns, financial data mining**

I. Introduction

Sequential data, often with timestamps, is increasingly common in applications, but most widely-used machine learning algorithms, particularly those that find *interpretable* models, are not designed to use it. Recently there have been several novel approaches to mining sequential data [10], [13]. Most work has been motivated by the problem of mining all frequent itemsets that occur with certain types of frequency, or periodicity. In this paper we are motivated by a different problem, namely to find all periods for certain kinds of itemsets. This is an issue which occurs in applications, e.g., in financial data, where it may be important to identify all recurring patterns in a transaction database. In this setting interpretable models are needed, e.g., due to regulatory constraints [5].

There are many methods that can be used to mine recurring patterns in sequential data (see [8] for a review). Such methods typically require a user-supplied parameter specifying the targeted periodicity, such as weekly, monthly, etc., for repeating occurrences of patterns. However, in applications where the focus is on *discovery* of periodic patterns, it may not be possible to specify ahead of time the target periods that are of potential interest. In this paper we describe a general data-driven approach where the length of period, or *periodicity*, of locally repeating patterns is estimated from data as part of the pattern mining algorithm. This avoids the requirement for the user to specify any target period in advance, and enables the discovery of more locally repeating patterns than existing approaches, with no significant reduction in efficiency.

In frequent pattern mining the principal measure is *frequency*, that is, the number (or proportion) of occurrences of a pattern in all transactions of the dataset. However, for a pattern occurring repeatedly in sequential data, this needs to be refined to capture a measure of its periodicity (if any) as well as the number of occurrences. In this paper the periodicity[1] of a pattern in a sequential dataset is based on: (i) the interval: a region of the sequence measured from left to right, such that within the interval the pattern occurs with a fixed (approximately) period; (ii) the period: the gap in timesteps between each occurrence of the pattern during the interval; and (iii) the support: the number of occurrences of the pattern during the interval — strictly this can be derived as the quotient of interval and period, but it is simple to compute it while finding patterns and it can be used to satisfy a constraint check on periodic patterns.

Since the period of a repeating pattern is essentially a first-order difference between consecutive occurrences of the pattern, we can use the second-order differences between consecutive periods to tell us something about the periodicity. In particular, if the second-order differences are small or close to zero over an interval that is large relative to the periods then the pattern is *locally repeating* [10] during that interval. Depending on the application, we can quantify values of interval or support that may be of interest. This implies a range of target periods, but does not require the user to specify a particular target period in advance.

The main contributions of this paper are: (i) we develop a new algorithm to identify locally repeating patterns without the requirement to specify a target period, (ii) we integrate it with the implementations of frequent pattern mining [7] presented by [10], (iii) we show on a real-world financial dataset and with controlled experiments on synthetic data that our new algorithm can find more patterns than the previous approach, and (iv) we show by comparison with the existing algorithms on two benchmark datasets that it does so without reductions in efficiency.

The remainder of the paper is organised as follows. In Section II we present relevant background, the framework is

[1]This overloads the normal use of the word "periodicity" which is typically a predicate – some behaviour is either periodic or not – since we add a quantification of the periodic behaviour in terms of its interval and period. We could have used the term "frequency" – as in a behaviour with frequency of 0.1 Hz occurs once every 10 seconds – but frequency already has a different meaning in the frequent pattern mining literature.

in Section III, the new algorithm is in Section IV, and the experimental evaluation is in Section V. We summarise our results in Section VI and conclude in Section VII.

II. Related work

We review several of the most common approaches, focusing on those that can generate interpretable (explainable) models, so we do not consider deep learning, e.g., [24]. In general, there is a separation of approaches applied to sequential data into two main areas: approaches in the first area tend to assume real-valued (continuous) variables, and data that is sampled at high-density or at regular frequencies (for example, data from sensors, stock-price data, and so on), whereas in the second area approaches can be characterised by typically dealing with discrete (Boolean, or nominal) variables, where data instances are ordered, although not necessarily timestamped. In the second area the problem of finding "useful" and "interesting" sequential patterns in data is typically framed as the discrete data version of time series analysis [1]. That is, the typical numerical values in time series are in sequential pattern mining replaced by *discrete* values, such as those denoting sets of items purchased in a retail transaction.

Sequential data was addressed early in research on frequent pattern mining [2]. Typically algorithms assumed market basket data and extended frequent itemset mining with refinements for sequential data, for example, to find sequences of itemsets that have support above some threshold in a dataset. Subsequently many extensions and refinements appeared in the literature; an excellent survey is [8]. In the literature the terminology of items and events came to be used interchangeably, so an itemset may mean a typical basket of items, or a set of events, occurring over a time interval. Algorithms can therefore be applied beyond market basket data to essentially any discrete, sequential data from protein sequences to timestamped financial data. Sequential pattern mining research continued to relax the definition of pattern and periodicity. For example, the time segments within which the periodicity is defined for a particular activity or event types may be dynamic, as in the record of an animal's behaviour [18]. Other variations of algorithms include time and memory efficient periodic pattern mining [21], [22], variance application over the period distribution in the database [6] and addressing the periodic interestingness of patterns [16].

The most relevant algorithms for the work in this paper are those that: (i) process atomic events, rather than itemsets (referred to as *event oriented patterns* in [20]); and (ii) include timestamp values in their calculations. Many algorithms, e.g., [14], [23] assume temporal databases or transaction databases where transactions are sequentially ordered, but do not have real-valued timestamps. However, we also consider algorithms for recurring pattern discovery and pattern change detection if they can be easily extended to include any of the above features.

The latest sequential pattern mining methods in the literature can be broadly divided into two groups: (i) algorithms that include real time, but where the pattern sequence is fixed, i.e., missing events are not permitted, e.g., [10]; and (ii) algorithms that do not consider an actual time-scale, but where patterns may contain "wild cards" to indicate missing events, e.g., [13]. Our work will follow [10] who define a periodic pattern over a sequence of transactions, each of which is an itemset with a timestamp.

Importantly, in [10] periodic patterns are identified *locally*, as they are not required to hold for the full duration of the dataset, but have start and end points defined by repeated occurrences with a period less than a user-supplied "maximal period" parameter. Local patterns can be found for any itemset by this approach, which can then be invoked within sequential frequent pattern mining algorithms adopting different search strategies such as breadth or depth-first to find all patterns above a minimum support threshold [10]. Our method will adopt the same approach as [10] but without the need to specify the maximum period. In [9] the TSPIN algorithm introduces the concept of the *lability* of periodic patterns, but this also requires the specification of the maximal period.

The technique we introduce in this paper to identify periodicity based on the differences between consecutive patterns is related to the second-order approach of [3], although in that work the goal is to identify anomalies based on contrastive patterns, rather than locally recurring patterns.

III. Terminology and definitions

In this section we describe the main concepts and representation used for finding recurrent (locally periodic) patterns. We assume a finite set of items U, where an *item* is typically a discrete symbol representing the occurrence of some event in data, such as the purchase by a user of some product, the membership of a cluster for some real value, or simply the appearance of some variable-value pair[2] A *transaction* is a pair (T_i, X_i), where T_i is the transaction *index* and $X_i \subseteq U$ is an *itemset*[3]. For an itemset Y, if the cardinality $|Y| = k$, we say that Y is a k-itemset. We assume indices are totally ordered. Specifically, in this paper all indices are assumed to denote the time of occurrence of the transaction, for some unit of time (e.g., microseconds, days, etc.). A *sequential* database D is a sequence of n transactions ordered by indices T_i, for $1 \leq i \leq n$. In this paper we will use *dataset* interchangeably with database. Unless mentioned otherwise, we will adopt the definitions for periodic frequent pattern mining from [10].

IV. An Algorithm to Mine Periodic Patterns

A periodic pattern for itemset I is a tuple (P, L, R, S), where P is the *base period* of the periodic pattern, L and R are the start and end timestamps (or transaction indices) denoting the time-interval over which the pattern occurs periodically in the dataset, and support S is the number of occurrences of I in the periodic pattern[4]. A periodic pattern for itemset I occurs in a sequence of S transactions (T_i, X_i) with $I \subseteq X_i$ for $1 \leq i \leq S$ with $T_1 = L$ and $T_{|S|} = R$. In [10] a

[2]Terminology in this paper is based mainly on that of [1], [10], [19].

[3]In this paper multiple items in a transaction are only counted once.

[4]Support may be omitted since it can be derived from P, L, R; see Section I.

(local) periodic pattern is defined according to the parameters determining the start and the end of the pattern: maximal period (maxPer), maximum spillover (maxSoPer), minimum duration (minDur) and minimum support (minSup). We also use these parameters, except for maxPer. Itemset I (which can be a single item or event) occurs in the dataset at consecutive transactions (T_i, X_i), (T_{i+1}, X_{i+1}) supporting I separated by a $period = T_{i+1} - T_i$, which must be less than a user-defined maximal period maxPer. Although not explicitly defined in [10] an event is understood to be a single item transaction. The difference $period - maxPer$ defines a *surplus* for each transaction, which can be positive or negative. The running (cumulative) sum of surpluses over time is called the $spillover$. The spillover is always non-negative, which is achieved by setting it according to $max(0, spillover)$. The time-interval of a pattern is defined by L and R, the start-point and end-point of the pattern. The start-point is any timestamp where $surplus \leq 0$. The end-point of the pattern occurs at a timestamp when spillover exceeds some user-specified maximum value, i.e., $spillover > maxSoPer$. The general sequence mining algorithm scans the sequential database for the starts and ends of any periodic patterns for an itemset. After the end of the previous pattern, the algorithm looks for the next transaction to satisfy the start-point conditions for the next pattern. An additional parameter, minimum duration ($minDur$), is introduced to constrain the time interval of the pattern, discarding patterns that are too short in time.

With the general approach described above, in [10] three methods, LPPM-breadth, LPPM-depth, LPP-Growth, define *how* the search for patterns is actually performed. All three methods use timestamp sets, replacing the TID sets in Apriori [2]. Given the parameters described above, the algorithms start with all single item itemsets, and expand each of them recursively, applying pruning where possible. The end product is of time-intervals that specify the intervals of all locally periodic patterns by their start and end points. LPPM-breadth expands itemsets keeping track of all partial itemsets, thus consuming more memory than LPPT-depth which expands each itemset until the branch is fully explored.

LPP-Growth is inspired by the FP-Growth algorithm [12] but is substantially modified to handle timestamps. In the first step, a prefix tree, here called LPP-tree, is constructed from the transaction database, then, in the second step, periodic patterns are mined recursively without scanning the original database. Similarly to the LPP-breadth and LPP-depth algorithms, LPP-Growth can be greatly simplified if the transaction is reduced to a single event.

Two types of Local Periodic Pattern (LPP) mining, the LPPM Breadth and Depth algorithms, are proposed in [10] by combining two mining strategies. The first uses only the Timestamp of Single Items (OTS strategy) to create periodic itemsets by intersecting single item timestamps. The second strategy was to map all itemsets with the same prefix (SPM) to the same key. Theoretical considerations and experimental results in [10] demonstrated better memory and runtime performance of LPPMBreadth2, which uses both strategies. In the case of LPPMDepth, a better performing version, LPPMDepth2, did not implement OTS to avoid generating the same prefixes multiple times. For these reasons, in this paper we base our algorithms and experimental evaluation on LPPMBreadth2 and LPPMDepth2, while LPPGrowth is unaffected by these considerations.

A. Algorithm AllPat

We propose a general algorithm to identify from an input sequential dataset the locally periodic patterns for a given itemset I, without the need to specify the target maximal period maxPer. To implement an algorithm that is able to identify all locally periodic patterns it turns out that it suffices to change the "$time2interval$" procedure of [10]. These changes result in the algorithm AllPat shown as Algorithm 1. Since this algorithm is the common procedure to identify local periodic patterns in each of the three algorithms developed in [10] we are able to adapt each of these algorithms to find all periodic patterns. The advantage of our proposed algorithm is that all local periodic patterns can be identified without the need for a used to specify maxPer, since it is not needed in AllPat.

We assume in Algorithm 1 that the input is an ordered list of timestamps for an occurrence of a given itemset (note that in actual implementations this is typically represented using a data-structure such as a bitset for efficiency [10]). Essentially the AllPat algorithm processes this list, which starts from the earliest timestamp, searching for a stable periodic pattern of occurrence of the itemset. AllPat does not require maxPer because it maintains a list of periods for the current pattern and updates the current period (curPer) as a function of the periods on this list. In Algorithm 1 curPer is updated at line 16.

Different methods can be used to calculate curPer from this list of periods; we have tested using either the most common period (mode) and the mean, and both can give good results, depending on the domain. We also implemented a version of the algorithm that determines curPer in terms a set of *target* periods such as weekly, monthly, quarterly, etc. This version of the algorithm is called TargPat.

We can view $maxSoPer$ as a *tolerance* parameter, i.e., by how much any individual period is allowed to exceed the periodicity of an identified pattern in the sequence, which is the spillover ($soPer$) of the currently open pattern. This is tested at line 9. Here $maxSoPer$ is specified as an amount of time, but it could be as a percentage number of $curPer$. If the spillover exceeds $maxSoPer$ the end-point of the pattern has been found, and the interval and base period are added to the list of patterns at line 11.

The end of the sequence can be handled in different ways; we essentially follow the method of [10], from line 21.

B. Example of the proposed algorithm

To illustrate the process of mining and to show differences between the existing LPPM and our new algorithm AllPat, an example dataset (Table I), similar to the transaction set in Table 1 of [10], is used to run both versions of algorithms. Since the final results are identical for LPPM Breadth, Depth

```
Algorithm AllPat(L,Q,R,S)
    Input: List $L$ of timestamps for transactions containing itemset $I$, minimum duration $minDur$, tolerance $maxSoPer$, minimum support $minSup$, last time point in dataset $largestTS$.
    Output: List of locally periodic patterns $P$.
    Initialise: $left = -1$, $t_0 = L[0]$, $t_1 = L[1]$, $curPer = t_1 - t_0$, $soPer = 0.0$, $sup = 0$, $P = \emptyset$.
    for $1 \leq i < |L|$ do
        /* find left endpoint */
        if $left = -1$ then
            $left = t_0$, $curPer = t_1 - t_0$,
            $sup = 0$, $soPer = 0.0$
        /* find right endpoint */
        if $left \neq -1$ then
            $soPer = soPer + t_1 - t_0 - curPer$
            if $|soPer| > maxSoPer$ then
                /* check duration */
                if $t_0 - left \geq minDur \wedge sup \geq minSup$ then
                    $P = \text{P.add}(curPer, left, t_0, sup)$
                $left = -1$
                $sup = 0$
                /* back up to last period */
                $t_1 = t_0$
            else
                $curPer = \text{curPer.get}(t_0, t_1)$
        $t_0 = t_1$
        $i = i + 1$
        $t_1 = L[i]$
        $sup = sup + 1$
    /* add final time point */
    if $left \neq -1$ then
        $soPer = soPer + largestTS - t_0 - curPer$
        if $|soPer| > maxSoPer$ then
            if $t_0 - left \geq minDur \wedge sup \geq minSup$ then
                $P = \text{P.add}(curPer, left, t_0, sup)$
        else
            if $largestTS - left \geq minDur \wedge sup \geq minSup$ then
                if $largestTS \neq t_0$ then
                    $sup = sup + 1$
                $P = \text{P.add}(curPer, left, largestTS, sup)$
    return $P$
```

Algorithm 1: A general algorithm for identifying locally periodic itemset patterns without the need for a target period. See text for details.

and Growth, in this example we only use LPPMDepth2 and LPPMDepth2_allPat, which is our extended version. In the example, the following parameters are used:

LPPM: minDur=4, maxPer=2, maxSoPer=2

AllPat: minDur=4, maxPer=10, maxSoPer=2

Since AllPat does not use maxPer, this parameter is set to the maximal timestamp value. We show only results for item "a" as this suffices to illustrate the process.

Patterns mined for "a" from Table I data:

LPPM: period 2, interval 1 - 13.

AllPat: period 2, interval 1 - 10; period 3, interval 10 - 18.

TABLE I
AN EXAMPLE OF A SEQUENTIAL TRANSACTION DATASET

Items	Timestamp	ts	Gap	soPer LPPM	curPer AllPat	curPer AllPat
a,b,c,e	6/06/2018	1	0	start,2	0	start,0
a,b,c,d	7/06/2018	2	1	1	1	0
a,b,e	9/06/2018	4	2	1	1	1
a,c,e	10/06/2018	5	1	0	2	0
a,b,d,e	12/06/2018	7	2	0	1	1
a,b,c,e	15/06/2018	10	3	1	2	2,end start,0
b,c,d,e	18/06/2018	13	3	2,end	1,3	1
a,c	22/06/2018	17	4	start,2	3	1
a,b,d	23/06/2018	18	1	1	4	-2,end start,0
a,b	25/06/2018	20	2	1	3,2	0

LPPM starts at timestamp index ts1 with soPer=2. After checking maxPer at ts17 soPer becomes 4, so the current pattern ends at ts13 and the algorithm starts looking for the next pattern at ts17. It cannot start at ts13, because the ts13 -ts1 interval exceeds maxSoPer. Since at the last timestamp, ts20, ts20-ts17 < minDur, LPPM discards the pattern. AllPat also starts with ts1, but with initial soPer=0. At ts13 soPer=3, therefore the current pattern ends at ts10. Initialising soPer=0 allows AllPat to start the next pattern at the same timestamp where the last pattern ended, ts10. Since at ts20 soPer becomes -3, the second pattern ends at ts18 and the third pattern starts at the same timestamp. The third pattern is not saved as as the last ts20 has duration < minDur. In summary, LPPM finds one pattern and cannot include the ts13-ts17 interval in the pattern. AllPat on the other hand covers the whole transaction set from ts1 to ts ts18, and breaks down the transaction set interval into two patterns.

V. Experimental evaluation

Our approach to evaluation in this paper is based on the following observation [17]: "In general, the most suitable evaluation criteria for a given problem tends to be domain-specific. That is, the evaluation criteria depends on what type of sequential patterns are of most interest in the domain application. Thus, the most general benchmark should provide a set of target patterns and evaluation criteria that is appropriate for each target pattern. Then, users can select the appropriate target patterns and evaluation criteria required for the domain in order to determine the most appropriate mining method for the application."

In this section we cover experimental evaluation of our method on (i) an open-source financial dataset, (ii) synthetic datasets for controlled experiments to assess the ability of our algorithm to recover known patterns, and (iii) compare it to the principal methods in [10] on two benchmark datasets.

A. Datasets

Performance of the proposed algorithms was evaluated on four datasets (Table II). The first three datasets contain real-world data with various characteristics, frequently used for benchmarking. The "Berka" and the "Artificial" datasets were selected to specifically evaluate algorithms on our domain of interest, financial transactions. Two datasets, "Kosarak" and "Online retail" are used as benchmarks for sequential pattern mining. The latter is a relatively small but dense dataset, measured by the number of items per record, while while the other three datasets are much sparser, containing about a million records each. Since the first three datasets represent an unsupervised learning task, we are mainly interested in runtime performance. The fourth dataset is artificially generated with known patterns as ground truth, with added noise, designed to evaluate the ability of different algorithms to recover true patterns, evaluated in terms of precision and recall. The datasets are described in more detail in the following subsections.

TABLE II
CHARACTERISTICS OF THE DATASETS

Dataset	#transactions	#items	#items per record
Berka	1,056,320	26,435	1.3
Kosarak	990,002	41,270	8.0
Online retail	23,260	4,224	22.7
Artificial	1,000,000	1	1.0

1) Financial transaction dataset – Berka Bank: The Berka dataset[5] contains information on anonymised transactions of a Czech bank, consisting of 4,500 accounts and about a million transactions. The transaction sequences exhibit a variety of periodic patterns, such as weekly, monthly and quarterly, as well as irregular, noisy instances. These transactions contain also irregular and unexpected patterns, for example bi-monthly, apparently more common in Europe. This is where our algorithms are expected to find meaningful patterns, without requiring a target period parameter setting.

Each Berka transaction contains 7 columns: *account_id, date, type, operation, amount, balance, k_symbol*. An example is: *2177, 930105, CREDIT, COLLECTION FROM ANOTHER BANK, 5123, 5923, OLD AGE PENSION*. For each account, we convert all transactions to the LPPM format [7] as "itemset|ts"; e.g., "1 2|5", where 1 and 2 are items in the itemset, and 5 is the timestamp. Each unique combination of *type*, *operation* and *k_symbol* is converted to a numeric item identifier, and *date* to timestamp in days starting from 1. If more than one transaction is on the same date, each of them becomes one item in the same itemset.

2) Benchmark datasets – Online Retail and Kosarak: This Online Retail dataset contains transactions for an online retail company recorded between 2009 and 2011. The company sells gift-ware mainly to wholesale customers. The data, originally from the UCI repository, has been pre-processed for evaluation of algorithms on sequences of itemsets. The Kosarak dataset [4] was created from sequences of click-stream data from a Hungarian news portal, pre-processed and converted to SPMF format. Both datasets are available for download[6].

[5]Available from: http://relational.fit.cvut.cz/dataset/Financial
[6]Available from: http://www.philippe-fournier-viger.com/spmf

3) Artificial dataset generation for controlled experiments: We defined time periods, such as 7, 14, 30, and 90 days, that are the most frequently observed based on an analysis of real-world financial transaction datasets. The dataset generation algorithm selects uniformly at random one of these intervals, called p, and generates n transactions containing the pattern with an interval period of $(p \pm delta)$. Here, n is an integer, between 1 and 24, selected uniformly at random. A user-defined value is chosen for $delta$ which can be expressed as an absolute number of days (e.g., 2 days) or as a percentage of days (e.g. 17% of 14 days). Such variations are frequently observed in real datasets. For instance, a customer may typically pay a fortnightly payment every 14 days, but occasionally after 15 or 13 days. Assuming a normal distribution with mean p, and using $delta$ as the standard deviation, random time interval values are generated. The above process is repeated to generate the required number of n transactions for each transaction sequence.

For this paper we used the data generator to produce six versions of the Artificial dataset with parameters reflecting characteristics of banking transactions. Three of these datasets were generated with an absolute value of delta, whereas the remaining three are generated with a relative delta, expressed as a percentage of the time interval between, e.g., 7, 14, 30, or 90. The idea is to make the standard deviations 1/3, 2/3, and 3/3 equal to the observed variations (i.e., 2 absolute days or 23% of interval days) in the customer transactions. As a result, the standard deviations for the three datasets for 2 absolute days are 0.67 (=2*1/3), 1.33 (=2*2/3), and 2.0 (=2*3/3), respectively, while the standard deviations for 23% of interval days are 7.7% (=23*1/3), 15.3% (=23*2/3), and 23% (=23*3/3).

Each dataset contains one million records, with the four intervals (7, 14, 30 and 90) in approximately equal proportion. Each interval, with delta applied, is repeated up to 24 times to make a (noisy) periodic pattern.

B. Evaluation methodology

The goal of the evaluation was to compare time performance and memory consumption of our extended algorithms with the original ones in [10]. Similar to [10], we varied parameters for each dataset, while trying to match the parameters for Kosarak and Online retail used in [10]. For some parameters, however, this was not always possible, mainly due to the fact that in our methods the spillover is calculated over local, dynamically found periods, whereas the original algorithms base their calculations on a globally set maximal period (maxPer). This makes our pattern intervals much shorter. Hence minimal duration also needed to be shorter. For this reason, the total number of itemsets and patterns would be too large to for the algorithm to complete in reasonable time. Therefore, we stop testing when the number of itemsets with periodic patterns found by each algorithm reached 10 million. We consider this methodology the most fair for both the LPPM and AllPat algorithms.

For Berka and Artificial datasets the chosen parameters are close to real-life banking transaction settings, based on domain knowledge. We conducted experiments with two maximal spillover (tolerance) values: (i) an absolute value of two days, which is a common number of days that banking customers deviate from their periodic transaction dates, and (ii) a tolerance of 23% of the current period, which is considered a maximal reasonable deviation. This makes for deviations of 2 days for weekly transactions, about one week for monthly, and about three weeks for quarterly transactions.

There is also one difference in parameter settings between LPPM and AllPat. The maximal period maxPer is not used in AllPat, as the spillover (tolerance) is based on the current period (see Section IV-A, whereas for LPPM this parameter is set by the user to an expected maximal period of any pattern.

All experiments were completed on an Intel Core i7 laptop with 32GB RAM, or a comparable desktop computer, with all experiments for a given dataset run on the same computer. Hardware configuration, however, is considered less important, since we compare only relative values.

C. *Runtime performance comparison*

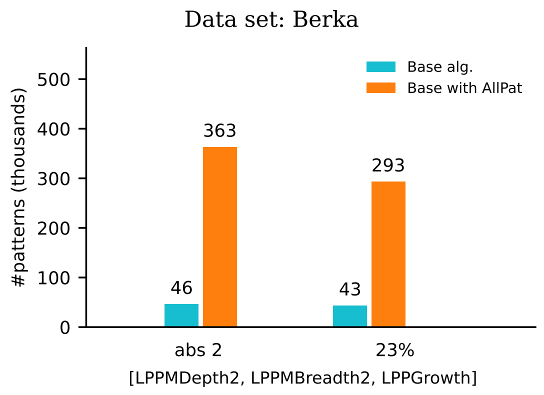

Fig. 1. Shown are performances in finding periodic patterns across all Berka accounts for the base algorithm compared to the extension finding all patterns. Clearly the new algorithm finds many periodic patterns *not* found by the base algorithm. Note that since *all* patterns are discovered when running each search strategy (depth, breadth or growth), the numbers of patterns found are the same. Two versions of the *tolerance*, i.e., $maxSoPer$ on variations in periodicity are shown. On the left, the results are for a value of $\pm$2days, whereas on the right these are for $\pm$23% of the current period ($curPer$).

Comparing time performance on all Berka accounts, a general observation is that AllPat finds approximately eight times more patterns than the base algorithms, with minimal time difference.

Both the absolute and 23% tolerance settings show similar trends and complete the mining in similar time. With the absolute tolerance, however, about 24% more patterns are found. This is because the larger tolerance leads to longer pattern intervals and patterns being merged.

For Online retail (Figure 2), the runtime and the number of patterns are similar when comparing each pair of algorithms. It is apparent, however, that both the extended and the baseline LPPMBreadth2 algorithms discover about four times more patterns in the same number of itemsets. These differences stem from the way the search for patterns works for different algorithms. Depth-first (DFS) and and breadth-first search (BFS) process single item itemsets first. After that, DFS combines items by progressively extending itemsets with the same prefix. More patterns are found initially for single items. BFS, on the other hand, processes more smaller itemsets first, so tends to find more patterns in the first 10 million itemsets. Since the Online retail dataset is relatively small (23,260 transactions), the algorithms manage to discover a large number of patterns in these smaller itemsets. However, the LPPMBreadth2 based algorithms are much slower for the same number of patterns. The explanation found in [10] is that the search space for small minDur and maxSoPer is much larger. LPPMBreadth2 groups items with the same prefix using a map, which reduces the memory usage but increases the processing time. The other two base algorithms do not use maps.

Similar results are obtained for the runtime on the Kosarak dataset (Figure 3), which is a very large and relatively dense dataset with over 41,000 distinct items. For similar reasons to Online Retail, the Kosarak dataset test results in Figure 3 show that LPPMBreadth2 algorithms discover more patterns but are about 2.5 times slower than LPPMDepth2, and an order of magnitude slower than LPPGrowth.

Comparing the base LPPM with AllPat, the latter tends to get larger number of patterns in shorter time. The differences, however, are very small. This is a desirable result, showing that the changes made to the base algorithms did not affect the runtime performance.

D. *Memory usage comparison*

Figure 4 shows memory consumption by the Java implementation of the algorithms on Online retail and Kosarak for the first 10 million itemsets. The Berka dataset is not included in this chart, as each account has to be processed separately, so the memory usage is minimal with a constant value of 5.12 Mb for all algorithms in our testing. The LPPMDepth and LPPGrowth pairs of algorithms (base and AllPat) use similar amounts of memory. The LPPMBreadth version uses much less memory at the cost of much higher processing time, as seen in Section V-C.

E. *Evaluation of pattern identification on Artificial datasets*

Here we investigate the ability of the algorithms to discover known periodic patterns from noisy data Since this is synthetic data we conducted controlled experiments and evaluated the number of patterns identified by the algorithm with different parameters.. We generated datasets as described in Section V-A3 and obtained test results for F1 scores.

In the following tests, AllPatand LPPMare evaluated on Artificial datasets with perfect periods of 7, 14, 30 and 90 days. The number of patterns for each period was approximately equal and randomly spread in the dataset. These periods commonly occur in real-time data. Only one LPPM algorithm, LPPMDepth2, was tested, since all base algorithms should

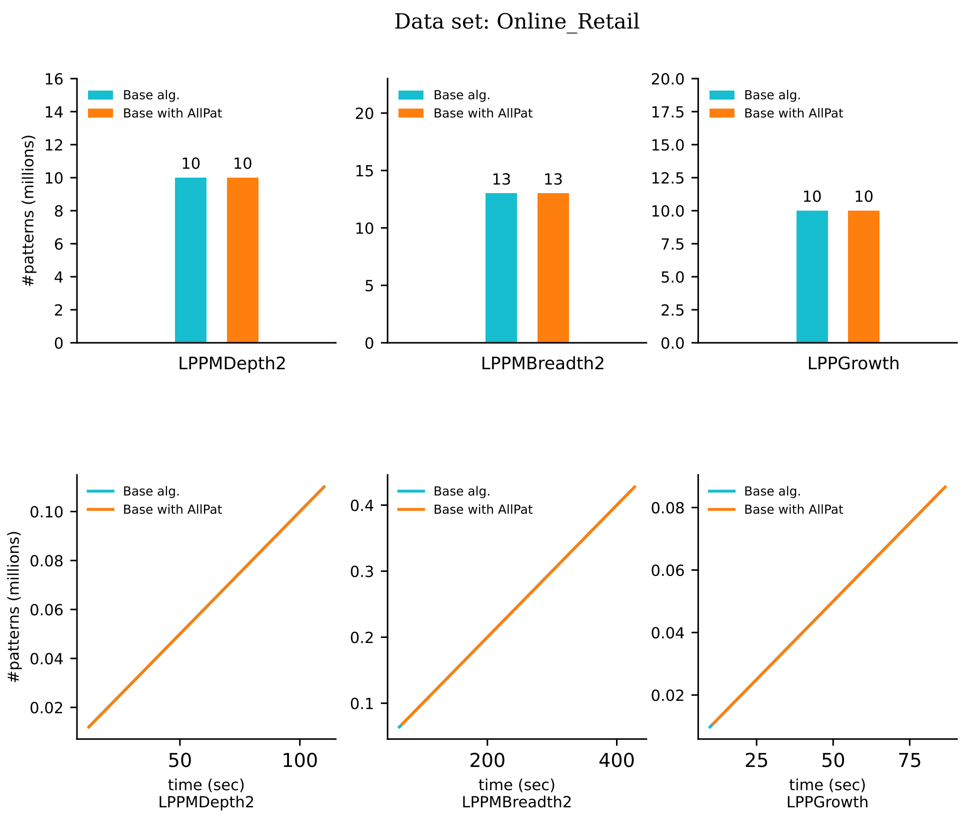

Fig. 2. Performance of base algorithm LPPM compared to AllPat extension on Online retail: maxSoPer=233 (1%), minDur=233 (1%), maxPer=1163 (5%).

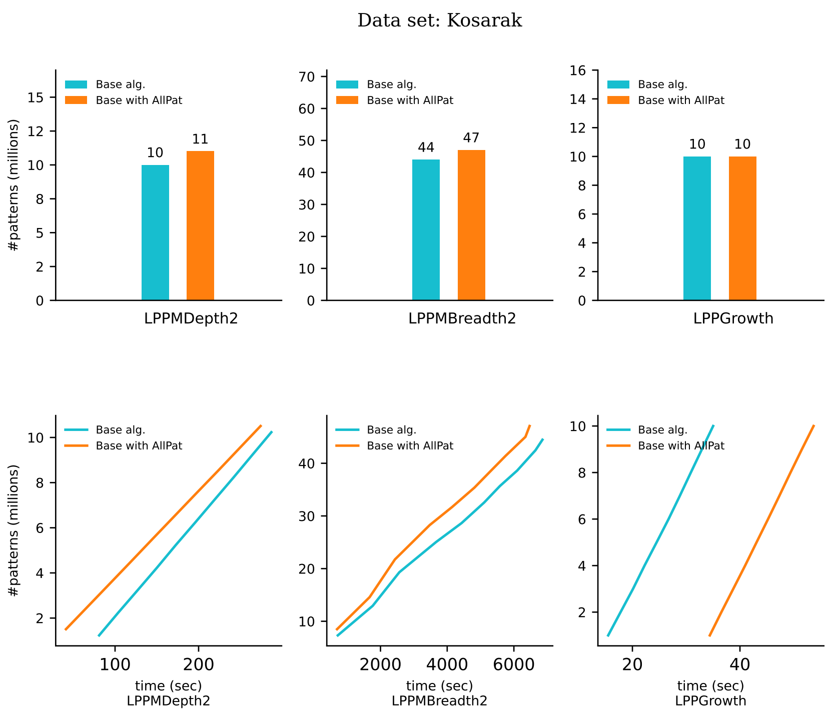

Fig. 3. Performance of base algorithm LPPM compared to AllPat extension on Kosarak: maxSoPer=1980 (0.2%), minDur=1980 (0.2%), maxper=1980 (0.2%).

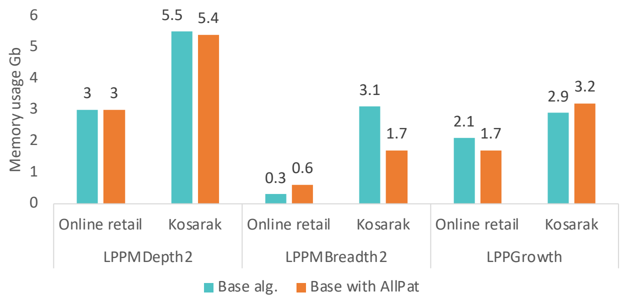

Fig. 4. Memory usage for Online retail and Kosarak datasets.

return the same results when run to completion. Four tests were conducted, one for each of the periods, where maxPer and minDur were set to the value of that period, and maxSoPer=1. In order to make the comparison possible, we added the same algorithm (described in Section IV-A) of nominating the base periods to LPPMDepth2 as they are used in AllPat. For the same sequence of periods in a periodic pattern, both LPPM and AllPat assign the same period, e.g., for the periodic sequence of timestamps 1, 8, 15, 20, the base period is 7.

TABLE III
F1 SCORE ON GENERATED PERFECT DATA (STDEV-0) LPPM: MAXPER=MINDUR=[7,14,30,90], MAXSOPER=1

	LPPM:7	LPPM:14	LPPM:30	LPPM:90
7 day	1	0.8233	0.6142	0.4047
14 day	0	0.8179	0.6107	0
30 day	0	0	0.6068	0
90 day	0	0	0	0
Micro F1	0.4047	0.5498	0.5243	0.2537
Macro F1	0.25	0.4103	0.4579	0.1012

In Table III, micro F1 scores are shown for each test and period, and the last two rows show micro and macro F1 for all periods. For AllPat, minDur was set to 7. The table does not show the results for AllPat, since the F1 scores are all 1.0, as expected.

For period 7 and maxPer=minDur=7, the base algorithm obtains a perfect F1 score, since 7 is the smallest period. For larger periods, the scores tend to get lower with higher period values, which can be explained by the way these algorithms combine separate periodic patterns into patterns with larger support up to the value of maxPer. Since the majority period is reported as the base period, for higher minDur, especially 90, the algorithm reports the majority period for a sequence that may combine patterns with different periods, as long as minDur is achieved. We note, however, that the base algorithms, in accordance with their design, do recognise almost all periodic patterns, although without defining their periodicity, as our extensions are designed to do.

Next, we show how well our algorithms discover 7, 14, 30 and 90 day patterns in the Artificial data, measured by the F1 score. Noisy, i.e., imperfectly periodic, time intervals were randomly selected as discussed in Section V-A3. In these tests we included only our algorithms, as the base algorithms are not designed to separate patterns by the length of periods. We include, however, our general algorithm AllPat and the TargPat version, specifically targeting these four periodic pattern.

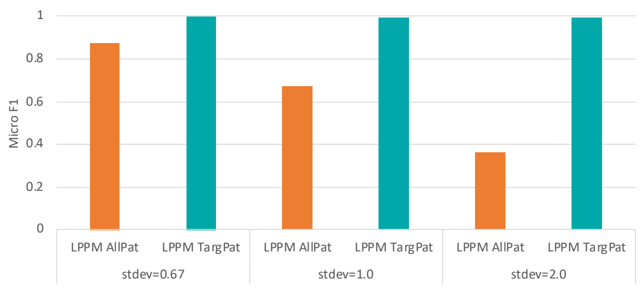

Fig. 5. F1 score for AllPat and TargPat with absolute *delta* and maxSoPer=2

In Figure 5, as expected, the F1 score decreases from 0.88 to 0.36 when the standard deviation increases from 0.67 to 2.0. The decrease is approximately the same for all four periods. The performance of TargPat, however is not affected by this amount of noise.

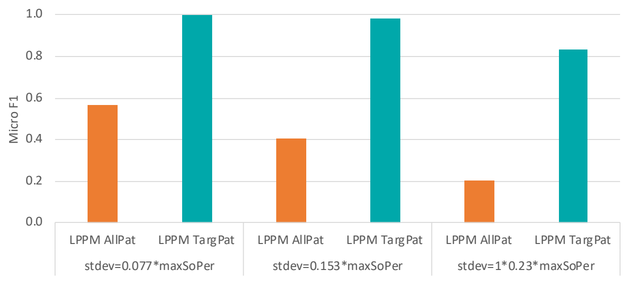

Fig. 6. F1 score for AllPat and TargPaton Artificial data with maxSoPer=23%

In the next chart (Figure 6), the data imperfection *delta* is increased to 1/3, 2/3 and 1.0 of 23% of current period interval, which is larger than in the case of absolute maxSoPer, as shown in Table IV, especially for the 90 period. The F1 score, in comparison with Figure 5, dropped by about 40%, while TargPat is almost unaffected.

In summary, TargPat is much more suitable when the periods of periodic patterns are known in advance. When we are interested in all patterns, AllPat is more suitable, although the accuracy may be lower.

TABLE IV
STANDARD DEVIATION APPLIED TO GENERATED DATA FOR EACH PERIOD

Per	maxSoPer	1/3*maxSoPer	2/3*maxSoPer	1*maxSoPer
7	1.6	0.5	1.1	1.6
14	3.2	1.1	2.1	3.2
30	6.9	2.3	4.6	6.9
90	20.7	6.9	13.8	20.7
all	2	0.7	1.3	2

F. A qualitative evaluation on real-world transaction data

Since *interpretability* of periodic patterns is important, in this experiment we investigated both the numbers of patterns identified on the Berka dataset for periods around 7, 14, 30, 90, in addition to inspecting the patterns from a selected account for potential insight.

In Figure 1 we showed that for the Berka dataset AllPat discovers an order of magnitude more patterns than LPPM (46,000 *vs.* 363,000). Table V shows the number of itemsets and patterns discovered by LPPM and AllPat for all accounts of the Berka dataset.

TABLE V
NUMBER OF ITEMSETS AND PATTERNS FOR BERKA DATASET USING FOUR VALUES OF PARAMETERS

Algorithm	maxPer	minDur	maxSoPer	#Itemsets	#Patterns
LPPM	7	5	2	6622	38217
LPPM	14	12	2	6497	32507
LPPM	30	28	2	36487	261671
LPPM	90	88	2	33181	36238
AllPat	no limit	5	2	44046	427252

For LPPM, maxPer and minDur are selected to extract specific patterns for comparable results. For AllPat, the maxPer parameter is not used. The total number of patterns for LPPM is 368633, still less than AllPat, noting that, as discussed in Section V-E, many patterns may be duplicated.

Next, we analysed in more detail specific fragments of patterns obtained by running the algorithms on the Berka account (# 8261 in the dataset) with the largest number of deduplicated transactions (over 400), finding important differences between AllPat and LPPM specific to this domain.

TABLE VI
EXAMPLE CASES FROM MINING THE LARGEST BERKA ACCOUNT

Case	Alg.	Item	Pattern	TS From	TS To	#Trans.
1	LPPM	3	p30	33	63	1
		3	p30	125	155	1
	AllPat	3	p31	2	186	6
2	LPPM	0	p150	0	510	5
	TargPat	-	-	-	-	-
	AllPat	0	p150	210	510	3
3	LPPM	3	p30	125	155	3
			p30	186	216	3
	AllPat	3	p31	2	186	7
			p31	186	337	7
4	LPPM	13	p5	146	2042	129
	AllPat	13	p5	146	151	2
			p26	151	177	3
			p5	177	182	2
			p25	182	207	3
5	LPPM	10	p30	60	346	16
	AllPat	10	p30	60	150	4
			p7	150	157	2
			p23	157	180	2
			7 more			
6	LPPM	13	p5	146	151	2
			p5	177	182	2
			p5	207	212	2
	AllPat	13	p5	146	151	2
			p26	151	177	2
			p5	177	182	2
			p25	182	207	2
			p5	207	212	2

In Table VI the item codes used are translated to the following original banking transaction types:
0: *CREDIT, CREDIT IN CASH*
3: *CREDIT, COLLECTION FROM ANOTHER BANK*
10: *DEBIT, CASH WITHDRAWAl*
13: *DEBIT, REMITTANCE TO ANOTHER BANK*
The p<n> notation denotes a pattern with base period n, e.g., p31 has a period of 31 days; TS From and To denote the interval start and end-points, and #Trans is the number of transactions containing the pattern, i.e., its support.
Case 1: LPPM breaks the monthly pattern into two patterns, since the maxPer and maxSoPer parameters prevent discovery of both the 30 and 31 day periods in the monthly pattern. AllPat with the same parameters uses negative soPer credit to discover both periods plus additional ones not covered by LPPM.
Case 2: LPPM with minDur=180 concatenates p30, p180 and p150 together, reported as p150. TargPat does not discover p150 at all, since this period is not on the target list and is not expected. AllPat separated the three patterns and properly reports p150.
Case 3: LPPM misses interval 155-186, because at 155 soPer reaches maxSoPer. AllPat does not have the maxPer limitation and includes this interval in p31.
Case 4: LPPM with higher maxPer concatenates many patterns together, while AllPat separates the whole interval into four patterns.
Case 5: To assess a bank account it is important to identify any irregularities in regular transactions. LPPM shows one P30 pattern between ts 60 and 346. AllPat identifies regular monthly withdrawals between timestamps 60 and 150, and then reports a number of irregular periods from 7 to 30 days.
Case 6: Item 13 (*REMITTANCE TO ANOTHER BANK*) is very irregular. LPPM discovers only p5 with missing intervals, whereas AllPat shows a variety of patterns from p5 to p26 fully covering the 146-212 interval.

The above cases show benefits of using AllPat on real banking data, highlighted in IV: (i) AllPat tends to cover time intervals more completely since it uses negative soPer to dynamically adapt to slightly smaller and larger periods, (ii) it finds more detailed patterns that include irregularities, and (iii) it better adapts to slightly changing periods, e.g., for 30 and 31 days.

The above cases identify some atypical patterns found by AllPat, such as short patterns of 5 and 25 days. These patterns alone may not be interesting, but when analysed in combination with other patterns may provide useful information about the status of an account.

VI. DISCUSSION

In the previous section we presented results for runtime, and omitted results for memory, since this is mainly a factor of itemset size, and in this work we are mainly concerned with itemsets of (relatively small) bounded size.

While testing the AllPat algorithms, we discovered some interesting features of the algorithms, in addition to those reported in [10]. First, the Breadth-First Search algorithms examine smaller itemsets first, resulting in a much larger number of discovered patterns in the first stage of mining. Second, the number of patterns does not tell the whole story. One algorithm can find just one pattern repeated 1000 times, while another may find a large number of patterns, each with very low support. A more suitable metric is required to measure the quality of patterns in a given domain. Third, the base algorithms find a smaller number of patterns with larger support if the periods are smaller than maxPer. This is observed in the first two thousand Kosarak single item itemsets. If periods tend to frequently exceed maxPer, the base algorithms report more patterns with lower support, as observed in Kosarak results. For the AllPat algorithms, the differences between subsequent periods matter the most. If these differences are within maxSoPer tolerance, AllPat reports a smaller number of patterns with larger support. As the differences get larger, the opposite occurs. These differences between periods do not matter to the base algorithm, as long as they do not much exceed maxPer, i.e., they are within maxSoPer tolerance.

These characteristics of AllPat methods allow them to discover more fragmented patterns, but with more susceptibility to noise. To control this we implemented a minimal support parameter, but it was not used in the comparison tests for compatibility with the original algorithms. We have not addressed the issue of the "interestingness" of discovered patterns [11] in this paper. However, research in a recent book [15] addresses this issue and we will investigate it for further work.

VII. Conclusion

The AllPat method proposed in this paper extends the state-of-the-art LPPM algorithms by removing the requirement to specify in advance the target period. Changing to a data-driven method of identifying periodic behaviour enables greater flexibility in the patterns that can be identified in sequential data and provides users with more detail. Since the new method is fully compatible with the main search strategies implemented by [10] these improvements retain the efficiency of the base algorithms, as we have shown empirically. When tested on the task of recovering known periodic patterns from noisy data our method performs well. On real-world data we showed the method's ability to find periodic patterns that the base algorithms miss, and with more detail on the periodicity that can be a potentially useful source of insight. Further work on methods for pattern interestingness may help to manage the greater detail on periodicity found by our approach. For reproducibility, code and data will be online on publication.

References

[1] C. Aggarwal. *Data Mining: The Textbook.* Springer, 2015.

[2] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proceedings of the Eleventh International Conference on Data Engineering*, pages 3–14. IEEE, 1995.

[3] E. Alipourchavary, S. Erfani, and C. Leckie. Mining Rare Recurring Events in Network Traffic using Second Order Contrast Patterns. In *Proceedings of the 2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2021.

[4] A. Benson, R. Kumar, and A. Tomkins. A discrete choice model for subset selection. In *Proceedings of the Eleventh ACM international Conference on Web Search and Data Mining*.

[5] F. Doshi-Velez, M. Kortz, R. Budish, C. Bavitz, S. Gershman, D. O'Brien, S. Schieber, J. Waldo, D. Weinberger, and A. Wood. Accountability of AI Under the Law: The Role of Explanation. *arXiv preprint arXiv:1711.01134*, 2017.

[6] M. Rashid *et al.* Efficient mining regularly frequent patterns in transactional databases. In S.-G. Lee, Z. Peng, X. Zhou, Y.-S. Moon, R. Unland, and J. Yoo, editors, *Database Systems for Advanced Applications*, pages 258–271, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[7] P. Fournier-Viger, A. Gomariz, T. Gueniche, A. Soltani, C.-W. Wu, and V. Tseng. SPMF: A Java open-source pattern mining library. *Journal of Machine Learning Research*, 15:3389 – 3393, 2014.

[8] P. Fournier-Viger, J. Lin, R. Kiran, Y. Loh, and R. Thomas. A Survey of Sequential Pattern Mining. *Data Science and Pattern Recognition*, 1(1):54–77, 2017.

[9] P. Fournier-Viger, Y. Wang, P. Yang, J. Lin, U. Yun, and R. Kiran. TSPIN: mining top-k stable periodic patterns. *Applied Intelligence*, 52:6917–6938, 2022.

[10] P. Fournier-Viger, P. Yang, R. Kiran, S. Ventura, and J. Luna. Mining local periodic patterns in a discrete sequence. *Information Sciences*, 544:519–548, 2021.

[11] J. Han, M. Kamber, and J. Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2011.

[12] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. *ACM sigmod record*, 29(2):1–12, 2000.

[13] Jen-Wei Huang, Bijay Prasad Jaysawal, and Cheng-Chung Wang. Mining full, inner and tail periodic patterns with perfect, imperfect and asynchronous periodicity simultaneously. *Data Mining and Knowledge Discovery*, pages 1–33, 2021.

[14] Kuo-Yu Huang and Chia-Hui Chang. Asynchronous periodic patterns mining in temporal databases. In *Databases and applications*, pages 43–48, 2004.

[15] R. Kiran, A. Mondal, J. Lin, J. Luna, and P. Fournier-Viger. *Periodic Pattern Mining*. Springer, 2021.

[16] R. et al. Kiran. Discovering periodic patterns in non-uniform temporal databases. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 604–617. Springer, 2017.

[17] Hye-Chung Kum, Joong Hyuk Chang, and Wei Wang. Benchmarking the effectiveness of sequential pattern mining methods. *Data & Knowledge Engineering*, 60(1):30–50, 2007.

[18] Chih Lai, Nga T Nguyen, and Dwight E Nelson. Mining periodic patterns from floating and ambiguous time segments. In *2005 IEEE International Conference on Systems, Man and Cybernetics*, volume 4, pages 3622–3627. IEEE, 2005.

[19] S. Peng and A. Yamamoto. Improvement of Sequential Pattern Mining Based on (k, l)-Frequency and Generative Probability. In *The Japanese Society for Artificial Intelligence 111th Study Group on Basic Problems of Artificial Intelligence*, pages 46–51, 2020.

[20] Xingzhi Sun, Maria E Orlowska, and Xiaofang Zhou. Finding event-oriented patterns in long temporal sequences. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 15–26. Springer, 2003.

[21] A. Surana, U. Kiran, and P. Reddy. An efficient approach to mine periodic-frequent patterns in transactional databases. In L. Cao, J. Huang, J. Bailey, Y.-S. Koh, and J. Luo, editors, *New Frontiers in Applied Data Mining*, pages 254–266, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[22] S. Tanbeer, C. Ahmed, B.S. Jeong, and Y.C. Lee. Discovering periodic-frequent patterns in transactional databases. In T. Theeramunkong, B. Kijsirikul, N. Cercone, and T.B. Ho, editors, *Advances in Knowledge Discovery and Data Mining*, pages 242–253, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[23] Qian Wan and Aijun An. Discovering transitional patterns and their significant milestones in transaction databases. *IEEE Transactions on Knowledge and Data Engineering*, 21(12):1692–1707, 2009.

[24] A. Zoonozi, J-J. Kim, X. Li, and G. Cong. Periodic-CRN: A Convolutional Recurrent Model for Crowd Density Prediction with Recurring Periodic Patterns. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)*, pages 3732—3738, 2018.