


Efficiently Finding Cyclical Patterns on Twitter Considering the Inherent Spatio-temporal Attributes of Data


Claudio Gutiérrez-Soto

(Departamento de Sistemas de Información,
Universidad del Bío-Bío, Concepción, Chile

 <https://orcid.org/0000-0002-7704-6141>, cogutier@ubiobio.cl)

Patricio Galdames

(Facultad de Ingeniería, Arquitectura y Diseño
Universidad San Sebastián, Concepción, Chile

 <https://orcid.org/0000-0003-3051-2413>, patricio.galdames@uss.cl)

Daniel Navea

(Universidad del Bío-Bío, Concepción, Chile
daniel.navea1601@alumnos.ubiobio.cl)

Abstract: Social networks such as Twitter provide thousands of terabytes per day, which can be exploited to find relevant information. This relevant information is used to promote marketing strategies, analyze current political issues, and track market trends, to name a few examples. One instance of relevant information is finding cyclic behavior patterns (i.e., patterns that frequently repeat themselves over time) in the population. Because trending topics on Twitter change rapidly, efficient algorithms are required, especially when considering location and time (i.e., the specific location and time) during broadcasts. This article presents an efficient algorithm based on association rules to find cyclical patterns on Twitter, considering the inherent spatio-temporal attributes of data. Using a Hash Table enhances the efficiency of this algorithm, called HashCycle. Notably, HashCycle does not use minimum support and can detect patterns in a single run over a sequence. The processing times of HashCycle were compared to the Apriori (which is a well-known and widely used on diverse platforms) and Projection-based Partial Periodic Patterns (PPA) algorithms (which is one of the most efficient algorithms in terms of processing times). Empirical results from two spatio-temporal databases (a synthetic data set and one based on Twitter) show that HashCycle has more efficient processing times than two state-of-the-art algorithms: Apriori and PPA.

Keywords: Cyclical Patterns, Spatio-temporal Data, Social Networks, Algorithms

Categories: H.3.1, H.3.2, H.3.3, I.1.2, I.5.3

DOI: 10.3897/jucs.112523

1 Introduction

Data science is a powerful tool to discover hidden knowledge from several sources such as mobile applications [Rejeb et al., 2022], Internet of Things (IoT) [Molinaro and Orzes, 2022], Industry 4.0 [Bhattacharya et al., 2022], and Smart Cities [Kandt and Batty, 2021]. These applications involve sources such as Global Positioning Systems (GPS) [Li et al., 2019] and Geographic Information Systems (GIS) [Lü et al., 2019], which are

usually managed by Spatio-Temporal Databases (STDB). STDB stores and processes Spatio-temporal data. These databases are well-known because they incorporate the temporal dimension as an additional attribute, which provides an extra complexity to be processed. Because of its nature, STDB tends to be extremely large, and simple transactions such as insertions and updates become expensive in terms of processing time [Gaede and Günther, 1998]. As a result, traditional Data Mining techniques must be adapted to incorporate the temporal attribute. Existing studies have demonstrated that the analysis of Spatio-temporal databases can yield valuable patterns such as patterns of animal movement, disease response, weather analysis [Revesz, 2009], climate change [Li et al., 2011], road traffic [Nakata and Takeuchi, 2004], criminal activity or terrorist events [Bora et al., 2013], and human sociological behaviour in location-based social networks [Song et al., 2015]

Many patterns can be extracted from STDB, particularly sequential patterns where events or episodes occur over time in a specific sequence. Broadly speaking, there are many types of patterns; however, among the best-known patterns, we can point out periodic patterns. Periodic patterns correspond to those that appear with certain regularity within a sequence. These patterns occur in specific time periods. For instance, a periodic pattern corresponds to the schedule in which the bus “Line 1” arrives at Grand Central Station in Manhattan. This schedule can be given each hour from 6:00 to 18:00. From this baseline, a period can represent any unit of time, therefore a period can be an hour, a day, a week, and so on. On the other hand, cyclical patterns are an extension of periodic patterns, where the events that make up a pattern can be given by an interval of periods. For example, in the peak hours between 18:00 and 20:00 (i.e., this is the interval in which the events occur), there is most traffic on Central Park Avenue. Note that the pattern occurs in about 24 hours. Other interesting patterns could be given in the context of social networks. For instance, the release of tweets from some specific places in a repetitive way at a particular moment of the week (e.g., tweets sent from a pub on Fridays between 22:00-24:00). On the other side, one of the foremost applications is given over the OLAP, MOLAP, and ROLAP databases since it is workable to realize an analysis of trends over time. To illustrate this, suppose that someone wants to analyze the sales by weeks; once done, it is necessary to analyze the sales by month. This latter is well-known as granularity, where the analysis can be conducted using time intervals such as days, weeks, and months, to mention a few.

As mentioned above, aiming to find the patterns, events or episodes that occur over time outlined in a specific sequence, considering the order in which the events take place. Later, an events’ minimum threshold is given to determine patterns along with the length of the periods to be analyzed. Notably, in this paper, we addressed the search for cyclical patterns.

1.1 Problem Definition

Let $o(s', t)$ be a spatio-temporal object determined by a spatial location s' and a point in time t . An event e (i.e., a change in the object’s location or shape) associated with an object o_x , considering a location s'_m at time t_i , is represented by $e(o_x, t_i)$. Without loss of generality, we assume that the space in which the objects can be located is partitioned into a set of $n \times n$ disjoint cells of equal size. Following this logic, s'_m corresponds to one of the cells in the set. A sequence of localized events denoted as S_x (i.e., for object o_x) is the set of events that takes place over a time series τ , such as $\tau = \{t_1, \dots, t_n\}$,

and $t_i < t_{i+1}$. An association rule for o_x is given as $X \rightarrow Y$, such that $X, Y \subseteq S_x$ and $X \cap Y \neq \emptyset$. The support of X , denoted as $sup(X)$, is the number of events in S_x that contain X .

Incidentally, each t_i can be seen as an hour, day, week, or any other interval of time, and so on. Using this rationale, if t_i represents a day, then seven consecutive t_i depict a week.

Definition 1: An association rule over the sequence S_x has a cycle $c(l, t_b, l_b)$, if the association rule appears every l^{th} unit of times starting at time t_b , considering an occurrence interval of length l_b , such that l is the length of the cycle (i.e., l can be seen as the number of periods), and whose support outperforms a threshold called sup_{min} .

Now consider a pattern denoted as X , such as $X \subseteq S_x$. We say X is an p -periodic pattern if the length of X is p ; where p is called the period. A pattern occurs when a sequence of events meets the minimum support provided by the user. It should be noted that the user also provides the p value. For instance, Let $p = 3$ be over the sequence $S = \{\{e\}\{b\}\{c\}\{e\}\{d\}\{c\}\{e\}\{c\}\{c\}\}$. Accordingly, there are three possible subsequences with three events each in which the pattern $\{e\}\{*\}\{c\}$ (i.e., $\{*\}$ can be any event) can appear. This means that the $sup(\{e\}\{*\}\{c\})$ is equal to 1 because $\{e\}\{*\}\{c\}$ appears in all subsequences (e.g., if the minimum support were $1/3$, the pattern should occur at least once within the subsequences). Therefore, $\{e\}\{*\}\{c\}$ is a perfect periodic pattern (i.e., it appears exactly every three periods), while simultaneously being a partial pattern with period $p = 3$, as the second event is missing. Conversely, when a pattern is absent in the sequence, it is imperfect. Remarkably, a full periodic pattern or a partial pattern can be both perfect or imperfect.

Aiming to differentiate between cyclical patterns and periodic patterns, let us consider the following sequence: $\{a\}\{b\}\{c\}\{x\}\{y\}\{d\}\{e\}\{g\}\{x\}\{d\}$. When we set $p = 4$, it becomes infeasible to identify $\{x\}$ within any pattern, taking into consideration that a pattern must appear at least twice. However, if we establish a cycle composed of five periods (referred to as l , denoting the cycle length) and assume an interval of two events or periods, we can discern the subsequences $\{x\}\{y\}$ and $\{x\}\{d\}$. Hence, we can designate $\{x\}\{*\}$ as a partial cyclical pattern that repeats every five periods (where $l = 5$, $l_b = 2$, and $t_b = \{x\}$). In a broader context, it is important to note that a cyclical pattern aligns with a periodic pattern when $p = l = l_b$, and t_b can represent any event within the subsequence, even if the event reappears at the subsequent $+p$ position. Noteworthy, this latter form of cyclical pattern corresponds to a periodic cyclical pattern.

1.2 Contribution

The main contribution of this paper can be itemized as follows:

- A new algorithm called HashCycle, which is based on a hash table to detect cyclical patterns over spatio-temporal data. Different from other algorithms grounded on association rules HashCycle does not require minimum support.
- HashCycle's time complexity is presented in a context where time and space are considered on search patterns.

- A set of experiments that involve both synthetic and real data are exposed. The synthetic dataset is used to corroborate that the implemented algorithms are sound (i.e., it means that they find the periodic cyclical patterns on the dataset), while the real dataset corresponds to a subset of issued Tweets from New York. Using the Tweets dataset was feasible to find cyclical patterns.
- Empirical results show that HashCycle is more efficient than PPA and Apriori algorithms in terms of processing times, which are in line with their respective time complexities.

The remainder of this paper is organized as follows. Section 2, reviews related work for detecting sequential patterns on spatio-temporal databases. In section 3, we present our proposed method for detecting cycle patterns. Section 5 describes empirical results. Finally, conclusions are presented in section 6.

2 Related Work

In the literature, methods for sequential pattern mining can be categorized into three groups: (i) methods that rely on mathematical approaches, (ii) methods that search for significant patterns using machine learning techniques, and (iii) methods that are based on association rules.

Group (i) encompasses algorithms that assess circular autocorrelation using Fourier transform [Malode et al., 2015, Parthasarathy et al., 2006]. Khanna and Kasurkar [Saneep and Swapnil, 2015] address three types of periodicity (symbol periodicity, segment periodicity, and partial periodicity) by proposing corresponding variants of autocorrelation-based algorithms. These methods are robust against noise and efficient in extracting partial periodic patterns with unknown periods and no additional domain knowledge. However, they cannot guarantee the extraction of all periodic patterns satisfying given conditions (e.g., minimum support). Other algorithms in this category have been proposed to find cycles in time series: the rainflow counting algorithm [Endo et al., 1974] and CyDeTS [Dambrowski et al., 2012, Gupta, 2022]. These algorithms detect cycles and provide cycle lengths that allow predicting life cycles for mechanical engineering applications and stationary electrical storages. Drawbacks include unsuitability for real-time applications, inefficient memory use, and the exclusion of spatial information.

Group (ii) includes machine learning approaches that require an objective function or a training dataset to define “good” sequential patterns [Jamshed et al., 2020, Bunker et al., 2021]. A primary drawback of these machine learning methods is their impracticality when users lack sufficient domain knowledge for managing training and tuning stages. We discard studying this category as it implies training and tuning stages.

Finally, group (iii) methods involve techniques primarily based on association rule mining algorithms. Many of these techniques derive from the “Apriori” approach proposed by Agrawal and Srikant [Agrawal and Srikant, 1994]. Apriori explores frequent itemsets, pruning infrequent ones during the process to manage the number of item combinations, preventing explosion. While still popular [Tirumalasetty et al., 2015], applying Apriori to sequential pattern mining is impractical due to potentially examining 2^n candidates. Possible optimizations leverage inherent properties in periodic pattern

mining [Ozden et al., 1998]. These authors propose two algorithms: one called the sequential algorithm, based on Apriori, and the second one called the interleaved algorithm. According to the authors, the improvement of the second approach with respect to the first one is about 5%. These algorithms explore unnecessary candidates and are not tested on real data but on randomly generated data. In this synthetic data, events are manipulated and taken to binary space for pruning, something that does not happen in the real world. For these reasons, these algorithms have been discarded from implementation and comparison in this paper.

It is relevant to highlight that the most related works [Agrawal and Srikant, 1994] and [Ozden et al., 1998] are not designed to work in a spatiotemporal context. For spatiotemporal data mining, several approaches have been developed for essential applications like disease diffusion analysis [Gao et al., 2018], user activity analysis [Lv et al., 2013], and local trend discovery in social media [Ishida, 2010, Cheng and Wicks, 2014]. Other approaches dealing with spatial information include encoding spatial features as discrete symbols (assigning symbols reflecting semantic regions), treating them as continuous variables [Pillai et al., 2012, Pillai et al., 2013], and formulating them as a dynamic graph mining problem [Lahiri and Berger-Wolf, 2010]. Koylu [Koylu, 2019] proposes a framework to model and visualize the semantic and spatiotemporal evolution of topics in interpersonal communication on Twitter. This work uses term frequencies to indicate that a topic has a strong relationship with a term. Here, a user sets the time period for pattern analysis. The framework produces a series of word-topic matrices that represent the prominent words and their probabilities in each topic and a series of document-topic matrices that represent the prominence of topics and their probabilities in each document. Then, these word-topic and document-topic matrices are used to group similar topics, allowing the identification of topic chains that illustrate temporally consistent topics. Finally, the topic evolution and spatiotemporal patterns are linked in a web-based geovisual environment. Gutiérrez et al.'s approach [Gutiérrez-Soto et al., 2022] aims to search periodic patterns, not cyclical ones. It computes the frequencies of events in a sequence and based on this factor discards some irrelevant patterns. It does not use any specific data structure (for example a hash table). Applying this approach to searching cyclical patterns would require mining and pruning all possible cycle periods. Moreover, this algorithm will not find cycles if they are not periodic.

An essential direction in this field involves optimizing approaches through improved algorithms and data structures. Han et al. [Han et al., 1999] propose the max-subpattern hit set algorithm, built upon a tailored data structure called max-subpattern tree, for efficiently generating larger partial periodic patterns. Yang et al. [Yang et al., 2013] propose a projection-based partial periodic patterns algorithm (PPA), more efficient than the max-subpattern hit set algorithm and Apriori for discovering partial periodic patterns.

To our knowledge, few works have addressed an efficient search for cyclical patterns over STDB and often neglect simultaneous time and space cost reduction. This paper introduces a novel and efficient algorithm (HashCycle) for identifying cyclical patterns over STDB. The discrete symbol encoding approach enables the exploitation of insights gained from mining sequential periodic patterns. HashCycle, inspired by a hash table, detects patterns through collisions. It doesn't demand minimum support, detecting all patterns appearing at least twice in a single execution. Our proposed algorithm demonstrates efficiency and scalability using real Twitter data.

3 Algorithms

In this section, an example has been given to clarify the steps involved in each algorithm, considering the temporal aspect.

3.1 Apriori Algorithm

Apriori checks all subsets and returns all those who achieved the minimum support. To achieve this goal, Apriori first acquires all frequent itemsets of size 1; after that, all frequent itemsets of size 2, and so on. Note that Apriori was not designed to consider the temporal aspect of events in a sequence. Therefore, considering the temporal aspect has a relevant impact on its performance. Aiming to show this impact, an example is given. Consequently, the example string can be perceived as a time series with period four, such that a letter represents an event. It is essential to highlight that the events in square brackets denote that these can occur in parallel (i.e., an event characterizes a particular object considering the space and temporal aspect).

$$a \{b, c, d\} eda \{b, c\} adabcacdaabc$$

According to the periodicity previously indicated, the number of periods in this series is four:

$$a \{b, c, d\} ed - a \{b, c\} ad - abcd - aced - aabc \quad (1)$$

The first step finds the L_1 set. L_1 involves all those candidates L_1 frequent that achieve the minimum support. Hereafter, F_k will be used instead of L_k . It is relevant to mention that F_k keeps the positions of the events in the sequence. From now, F_k will be employed in all algorithms. In the following example, we use the notation F_{yCand} to denote the frequency of candidate events in forming a pattern composed of y events. Therefore, using minimum support of 3, we have:

$$F_{1Cand} : \left\{ \begin{array}{l} a*** : \frac{5}{5}, *a** : \frac{1}{5}, **a* : \frac{1}{5} \\ *b** : \frac{3}{5}, **b* : \frac{1}{5}, ***b : \frac{1}{5}, *c** : \frac{4}{5}, \\ **c* : \frac{2}{5}, ***c : \frac{1}{5}, *d** : \frac{1}{5}, ***d : \frac{3}{5} \end{array} \right\}$$

$$F_1 : \left\{ a*** : \frac{5}{5}, *b** : \frac{3}{5}, *c** : \frac{4}{5}, ***d : \frac{3}{5} \right\}$$

$$F_{2Cand} : \left\{ \begin{array}{l} ab** : \frac{3}{5}, ac** : \frac{3}{5}, a*d* : \frac{3}{5}, \\ *{b,c}** : \frac{2}{5}, *b*d : \frac{2}{5}, *c*d : \frac{3}{5} \end{array} \right\}$$

$$F_2 : \left\{ ab** : \frac{3}{5}, ac** : \frac{3}{5}, a*d* : \frac{3}{5}, *c*d : \frac{3}{5} \right\}$$

Following the same steps gave by Apriori, the algorithm finishes when $F_K : \emptyset$

$$F_{3Cand} : \{ a \{b, c\} ** : No, ab*d : No, ac*d : \frac{3}{5} \}$$

$$F_3 : \left\{ ac*d : \frac{3}{5} \right\}$$

Following this thread, the number of events in F_3 does not generate a set of F_4 candidates. Hence, the algorithm finishes with F_3 .

3.2 PPA Algorithm

The PPA algorithm works the following: first, it runs the sequence and divides it into l partial periods. Later, every event is codified considering its location within the partial period. This way, codified events are represented as a matrix, which the authors name as *EPSD*. In *EPSD*, the first row is determined by the first l codified events, such that each column represents the event position within the partial periods. Thus, it is possible to count the instances of each event by columns. This last step is to verify if events accomplish the required minimum support. To exemplify this, a particular instance of Eq(1) is employed, such as:

$$abed - abad - abcd - accd - aabc$$

$$EPSD = \begin{pmatrix} a_1 & b_2 & e_3 & d_4 \\ a_1 & b_2 & a_3 & d_4 \\ a_1 & b_2 & c_3 & d_4 \\ a_1 & c_2 & c_3 & d_4 \\ a_1 & a_2 & b_3 & c_4 \end{pmatrix}$$

where the element $x'_{j,i}$ in *EPSD* corresponds to the event x in the position i in the sequence, such that the first subsequence ($j = 1$) is the first row in *EPSD*, the second subsequence ($j = 2$) corresponds to the second row, and so on.

A candidate subsequence is obtained once those events are counted by columns and achieve the minimum support. Subsequently, events that form this sub-sequence are sorted first, considering the partial positions and then considering each event's lexicographic nomenclature (the maximum nomenclature has size a). Note that the last sub-sequence S_c is equivalent to having F_1 . According to the authors, S_c is used to look for the other F_{kCand} patterns. A sub-routine called *Finding-FTP* is used to achieve this goal, so each event of S_c is used as a prefix to obtain the patterns that comply with the minimum support over *EPSD*. Finally, all F_k that fulfil the minimum support are obtained.

3.3 HashCycle Algorithm

The underlying motivation for employing a Hash Table is to reduce processing times during insertion, particularly for pattern searching. Furthermore, the order in which events take place in the sequence (i.e., the possible periods) makes them easily located in the Hash Table slots. Once this is done (i.e., when the Hash Table contains all the events), patterns can be easily obtained by traversing the lists belonging to the slot where the events appear. The occurrence of these events maintains the order in the lists with a positional difference of p . It's important to mention that each slot can have multiple lists, where the first element in the list determines the appearance of the subsequent elements in that list. Indeed, the length of each list indicates whether the minimum support can be achieved. This last aspect makes the Hash Cycle extremely efficient when it comes to finding patterns. To achieve this goal, we utilize the modulo operator, which is widely used in Hash Tables, particularly when the data universe (i.e., keys) is numeric. The latter allows us to quickly locate the slot in which the event is. For example, calculating the modulus between $(4 \bmod 2) = 0$ corresponds to the remainder of the division between 4 and 2. The previous one places us in slot 0, where the insertion or search takes place. It

should be noted that computing the modulus operation takes $O(1)$ (see Line 4, where h is provided the modulus), which is very efficient. Notably, we use this operator in our algorithm. By the way, our algorithm's main disadvantage is the Table's size since the different periods imply different Tables. Nonetheless, this is not a disadvantage due to the memory capacity of conventional computers. The following paragraphs describe how our algorithm operates on the Hash Table.

HashCycle relies on a hash table; Table T contains m slots, which store the events; in turn, these m slots have other s slots (i.e., which can be at most p), each one such that each slot stores a list with candidate patterns, considering the position in which events appear in the sequence. In the beginning, HashCycle runs the sequence and checks if the event is in T , so whether the event is not in the table, the event is added in $T[e_j]$ (i.e., $1 \leq j \leq m$) considering at the same time the position in which the event occurs in the sequence. This latter determines the following slot s'_k , between 1 and p , storing the event in $T[e_j].s'_k$ (see line 4 of the Algorithm). On the contrary, if the event is in T , it is put in its corresponding list. Once the sequence has been run, it is workable to determine the patterns. To achieve this goal, the period has to be previously provided. Pattern cyclical are obtained from lists of each event; note that these are perfect cyclical patterns. F_1 is extracted by running all those lists having at least two elements. As regards F_2 , this is built using the combination of F_1 , i.e., combining all lists with two or more elements (many implementations can be found in several algorithms' books). Following this reasoning, F_k (it is achieved with Lines 2, 3, and 4 of the Algorithm) can be achieved in the same way. It should be noted that Algorithm 1 allows achieving all patterns, as it processes all slots of T along with their corresponding lists. Aiming to shed light suppose the following example given the following sequence $aac - aab - aa$. There are two lists in T (see Figure 1) with period three over a sequence formed by events a, b, and c, whose length is 8. A list contains the following events (a,1), (a, 4), (a, 7), meantime another list is composed of (a,2), (a,5), (a, 8), such that the first list can be seen as $T[a].s_1$, and the second one as $T[a].s_2$. On the other hand, suppose the following lists (a,1), (a, 4), (a, 7), and (b,2), (b,5), (b, 8), combining both is feasible to obtain the pattern (a, b,*).

Algorithm 1 HashCycle

Require: $sup_{min} \geq 0 \wedge S \neq \emptyset \wedge n \geq 0 \wedge m \geq 0 \wedge p \geq 0$

Ensure: $Patterns$

```

1:  $Patterns \leftarrow \emptyset$ 
2: for  $x \leftarrow 1, x \leq m$  do
3:   for  $i \leftarrow 1, i \leq n$  do
4:      $T[e_i].s_{k'} \leftarrow h(p, e_i(o_x, t_i))$ 
5:     if  $T[e_i].n_{k'} \geq sup_{min}$  then
6:        $T[e_i].n_{k'} \in Patterns$ 
7:     end if
8:   end for
9:   return  $Patterns$ 
10: end for

```

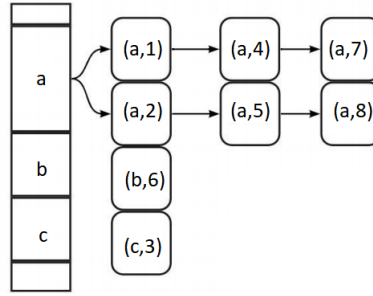


Figure 1: HashCycle's Data Structure

4 Time complexities

This section gives the time complexities for the algorithms Apriori, PPA, and HashCycle. Those time complexities are yielded in a Spatio-temporal context where the order in which the events occur plays a crucial role.

4.1 Apriori's time complexity

According to [Gutiérrez-Soto et al., 2022], Equation (2) represents all candidate patterns provided by Apriori such that m is the number of events, f indicates the size of a period, and m^f corresponds to the number of forms that can appear in the events. $C_{p,f}$ represent all ways to organize f events over a segment of size p . Owing to a pattern should appear at least twice over the sequence, and then it only is necessary to run the middle of the sequence; therefore, p can achieve $\frac{n}{2}$.

$$m^f * C_{p,f} = m^f * \binom{p}{f} = m^f * \frac{p!}{(p-f)! * f!} \quad (2)$$

The equation represents all candidate patterns for $f = 1$:

$$\sum_{p=2}^{n/2} m * p \quad (3)$$

Expression (3) can be implemented in any programming language using three loops. Recall that a pattern is considered as such if it appears at least twice within the sequence. In this manner, the outer loop runs from 1 to $\frac{n}{2}$ (i.e., $p = 2$ to $\frac{n}{2}$). This implies that half of the sequence could potentially form a pattern. The middle loop iterates through the sequence (i.e., it goes through n). These two loops provide $\frac{n^2}{2}$, which is equivalent to n^2 . The innermost cycle runs from 1 to m (i.e., such that m are the events). Therefore, the time complexity of (3) is determined by:

$$O(mn^2) \quad (4)$$

Extending Equation (2), it is workable to obtain all candidate patterns (i.e., all

candidates from F_{2Cand} to $F_{\frac{n}{2}Cand}$, thus:

$$\sum_{p=2}^{n/2} \sum_{F=1}^p m^F * \frac{m^F * p!}{(p-F)! * F!} \quad (5)$$

Note that the goal of Expression (5) is not to solve the expression itself (i.e., from a mathematical point of view), but instead to analyze how it can be implemented. It should be noted that many techniques can be used to calculate the factorial, such as dynamic programming. Due to this, the calculation is repeated many times. Therefore, the last one can be executed outside of both summations, making the use of each factorial a constant-time operation. On the other hand, it is evident that in the inner summation, $m^F * m^F$ appears, which from a programming perspective is essentially a multiplication by itself and operates in $O(1)$ time. Without loss of generality, expression (5) can be simplified as $n \sum_{F=1}^{n/2} m^F$, such that F takes the p value as maximum (see Expression (5)). From this latter expression, the n , which multiplies the summation, corresponds to a loop until n , while the summation itself is another cycle ranging from 1 to $\frac{n}{2}$. As a result, this last loop takes $O(m^{\frac{n}{2}})$. Therefore, the time complexity for Apriori is given by [Gutiérrez-Soto et al., 2022]:

$$O\left(n\sqrt{m^n}\right) \quad (6)$$

4.2 PPA's time complexity

Calculating the sub-sequence S_c is similar to computing F_1 ; Roughly, calculating the sub-sequence S_c is similar to computing F_1 , Whereby it takes $O(mn^2)$. Additionally, two previous sorts are necessary before obtaining F_1 . One must be computed for the partial position, while the second corresponds to the event's lexicographic nomenclature. The first sort takes $O(p \cdot \log_2 p)$; meantime second implies $O(a \cdot \log_2 a)$, where p corresponds to the period, and a is the maximum nomenclature of an event. Both sorts can be carried out using MergeSort when p and a are remarkable. Lastly, a subroutine builds all prefixes in S_c (i.e., for each event) and verifies them inside *EPSD*. Finally, the time complexity for PPA is given by [30] (if $((p \log_2 p) + (a \log_2 a)) < (m^{m+1} n^2)$):

$$O(\max((m^{m+1} n^2), (p \log_2 p) + (a \log_2 a))) = O(m^{m+1} n^2) \quad (7)$$

4.3 HashCycle's time complexity

Aiming to provide the time complexity for HashCycle, we introduce the following definitions:

Definition 2: Let $|S(\tau)|$ be a number of events over the sequence S .

Definition 3: Let $R\{e\}$ be the indicator random variable over a sequence S , which is associated with event e , such that:

$$R\{e(o_x, t_i)\} = \begin{cases} 1 & \text{if } e \text{ occurs at the } t_i \text{ moment over } S. \\ 0 & \text{if } e \text{ does not occur at the } t_i \text{ moment over } S. \end{cases}$$

Note that our sample space \mathbb{S} is formed by probability (H), it is that e happens at the t_i moment over S , and the probability (D), where e does not happen at the t_i moment

over S ; thus $Pr\{H\} = Pr\{D\} = \frac{1}{2}$. Therefore, the indicator random variable $R\{H\}$ for \mathbb{X}_H can be written as: $\mathbb{X}_H = R\{H\}$.

Lemma 1:

Given \mathbb{S} and an event $e(o_x, t_i)$ over S , let $\mathbb{X}_e = R\{e\}$.

Then $E[\mathbb{X}_e] = Pr\{e\}$.

Proof:

By the *Definition 3* and the definition of expected value:

$$\begin{aligned} E[\mathbb{X}_e] &= E[R\{e\}] \\ &= 1 \cdot Pr\{e\} + 0 \cdot Pr\{\bar{e}\} \\ &= Pr\{e\}. \end{aligned}$$

such that $Pr\{\bar{e}\}$ corresponds to the complement of e . ■

Definition 4: Let α be the load factor for a hash table T , such that $\alpha = \frac{n}{m'}$ corresponds to the average number of elements stored in a chain in T . T has m' slots that stores $|S(\tau)| = n$ occurrences of events. In our particular, case $m' = mp$ where m are events, and p is the period. In simple words, each event e_j has s slots, which are between 1 and p . The events are stored in T along with their position that appears in $S(\tau)$.

Definition 5: Let h the hash function over the hash table T , such that:

$h(p, e_j(o_x, t_i)) \rightarrow T[e_j].s_{k'}$, where $k' = (i \bmod p)$, and \bmod provides modulus or remainder of a division. Note that i determines the position of an event inside $S(\tau)$.

Definition 5: Let $n_{k'}$ the length of the list $T[e_j].s_{k'}$, so that n is

$$\sum_{j=1}^m \sum_{k'=1}^p T[e_j].n_{k'}$$

Theorem 1:

In T , an unsuccessful search takes $\Theta(1 + \alpha)$, under the assumption of simple uniform hashing.

Proof:

An event e_j that does not store yet in the table is equally likely to hash to any of the m' slots, assuming a simple uniform hashing. The expected time for a search unsuccessfully implies achieving the end of the list $T[h(p, e_j(o_x, t_i))].n_{k'}$, which has expected length $E[n_{h(p, e_j(o_x, t_i))}] = \alpha$. Therefore, the expected number of events reviewed in an unsuccessful search is α , and the total time required is $\Theta(1 + \alpha)$. ■

Theorem 2:

In T , a successful search can be solved in $\Theta(1 + \alpha)$, under the assumption of simple uniform hashing.

Proof:

Let e_j denote the event to be searched on T . Without loss of generality, it is workable to have the list from $T[e_j].s_{k'}$ (by *Definition 5*). As the distribution is uniform and the expected length of $T[e_j].n_{k'}$ corresponds to $E[n_{h(p, e_j(o_x, t_i))}] = \alpha$ (by *Definition*

4), then the search algorithm performs no more extended than $1 + \alpha$ comparisons (by Theorem 1). As a consequence, a successful search can be solved in $\Theta(1 + \alpha)$. ■

Consider that the time required to search an event depends on the length $n_{k'}$ of the list $T[e_j].s_{k'}$. Towards that end, the hash function $h(p, e_j(o_x, t_i))$ is computed in $O(1)$.

Incidentally, on this occasion, we have used the Θ notation since the worst-case is given when all the events are in the same list, by which, it should take $O(n)$. Nevertheless, we assume that the events follow a uniform probabilistic distribution; hence, the lists in the table are likely to have similar lengths. In this way, the Θ notation is more appropriate for this analysis.

5 Experiments

5.1 Experimental Environment and Empirical Results

Our experiments were carried out over two data types: synthetic and actual. Actual data corresponds to tweets. The algorithms were implemented in Java. To obtain correct measurements of processing times on JVM (Java Virtual Machine), JMH (Java Microbenchmark Harness) has been used. JMH is a Java framework used to build, run, and analyze benchmarks written in Java and other JVM languages. By doing this, it is possible to isolate the algorithms' processing times, ignoring elements such as caching, pipelines, and processing priority. JMH included ten training iterations and fifteen measurement iterations in all the experiments.

All the results are shown in average times (Avg)—times are expressed in milliseconds in Tables I-IV—along with its standard deviation (SDev), the latter to show the variability of the algorithms regarding the data. All experiments were executed in an Intel Core i3-5005U 2GHz; 8GB 1600 MHz DDR3L; and Windows Operating System 10 Home 64 bits.

Experimentation over synthetic data aims to check that all algorithms are sound (i.e., all algorithms find and return the same patterns). Two events' sequences have been created, such that the events occur randomly over the sequences (note that the events are known, and the uniform distribution has been used to allow them to occur in different places within the sequence. The first sequence does not contain patterns (see Table I). In the second sequence, a pattern of length ten has been added (see Table II). In both sequences, the experiments involve searching for patterns of lengths 10, 50, and 100 on sequences of lengths 500 and 1000.

Tables I and II show the average running times over a synthetic dataset. HashCycle presents the best performance – in terms of processing times — considering all scenarios (i.e., lengths, patterns, and sequences). Furthermore, HashCycle's standard deviations are the lowest, indicating that its behaviour is scalable independent of the pattern's length to search and the sequence's length. PPA achieves the second-best performance; nonetheless, it presents a higher standard deviation than HashCycle. Finally, the worst results are given by Apriori, where in some cases it outperforms 20000 milliseconds by which its results are omitted (the results appear as “-” in Table II).

Regarding real data, these involve four New York districts; Brooklyn, The Bronx, Manhattan, and Queens. Aiming to homogenize the number of inhabitants, we have built two groups: The Bronx - Brooklyn (designated as the L1 sector) and Manhattan - Queens (called the L2 sector). Incidentally, using the endpoints of a Twitter API, it is possible to identify the tweets issued from both sectors. The schedule used in the analysis of the tweets corresponds to prime time, which is between 21:00 and 21:30. To identify time ranges in which tweets are emitted, three periods, $\mathbb{T}_1=21:00-21:10$, $\mathbb{T}_2=21:10-21:20$, and $\mathbb{T}_3=21:20-21:30$, have been defined. Using sectors and periods, we can define a set of events; $\mathbb{E} = \{A, B, C, D, E, F\}$. It should be noted that the events associated with the first period (\mathbb{T}_1) only include events A and B , while the events linked to the second period (\mathbb{T}_2) are only composed of events C and D ; the same reasoning applies to the third period. The data collected comprised a week, starting on January 27th, 2021. Therefore, the sequences used by the three algorithms involve all the week's events. In this way, there are three sequences $S_{\mathbb{T}_1}$, $S_{\mathbb{T}_2}$, and $S_{\mathbb{T}_3}$ (i.e., the sequences associated with the periods \mathbb{T}_1 , \mathbb{T}_2 , and \mathbb{T}_3 , respectively) (see columns in Table 2). Finally, it is worth mentioning that in these experiments, lengths between 2 and 100 have been used to search for patterns, considering that all $S_{\mathbb{T}_i}$ comprise more than 1000 event occurrences each.

Tables III and IV display the average running times with actual data extracted from Twitter. Following the trending of Tables I and II, HashCycle presents the best processing times in all scenarios. Once again, HashCycle's standard deviations are the lowest. On the other hand, the second-best performances are accomplished by PPA; similarly to Tables I and II, PPA has higher standard deviations than HashCycle. Apriori provides the worst results.

In summary, HashCycle presents the best performances and provides the lowest standard deviations. It is worth stressing that running times present a significant saving regarding its adversaries. Finally, the second-best performance is achieved by PPA, followed by Apriori.

		Synthetic Without Patterns			
		500		1000	
Algorithm	Length	Avg	SDev	Avg	SDev
Apriori	10	0,790	0,009	1,529	0,009
Apriori	50	9,154	0,099	15,231	0,139
Apriori	100	35,727	0,329	55,294	0,424
PPA	10	0,249	0,003	0,523	0,006
PPA	50	0,547	0,007	1,637	0,009
PPA	100	0,791	0,005	2,372	0,016
HashCycle	10	0,017	0,001	0,038	0,001
HashCycle	50	0,031	0,001	0,065	0,001
HashCycle	100	0,047	0,001	0,099	0,002

Table 1: Experiments with synthetic data without patterns.

		Synthetic With Patterns			
		500		1000	
Algorithm	Length	Avg	SDev	Avg	SDev
Apriori	10	0,667	0,049	1,264	0,013
Apriori	50	1779,36	14,402	2548,39	63,68
Apriori	100	-	-	-	-
PPA	10	0,232	0,011	0,468	0,007
PPA	50	3,520	0,051	6,894	0,131
PPA	100	3745,8	161,01	6532,43	192,94
HashCycle	10	0,019	0,001	0,044	0,001
HasgCycle	50	0,043	0,001	0,068	0,001
HashCycle	100	0,071	0,001	0,144	0,009

Table 2: Experiments with synthetic data with patterns.

		S_{T_1}		S_{T_2}	
Algorithm	Length	Avg	SDev	Avg	SDev
Apriori	10	0,952	0,007	1,197	0,012
Apriori	50	9,822	1,500	10,888	0,332
Apriori	100	28,601	0,190	42,312	0,408
PPA	10	0,366	0,010	0,465	0,005
PPA	50	0,633	0,010	0,919	0,010
PPA	100	0,865	0,010	1,178	0,010
HashCycle	10	0,044	0,001	0,060	0,001
HasgCycle	50	0,113	0,008	0,134	0,005
HashCycle	100	0,171	0,002	0,224	0,008

Table 3: Experiments with data extracted from Twitter (S_{T_1} and S_{T_2}).

6 Discussion

As mentioned earlier, a cyclical pattern is equivalent to a periodic pattern if the cyclical pattern can be expressed as a periodic pattern (i.e., it is a periodic cyclical pattern). Based on this foundation, it is feasible to detect periodic cyclical patterns using those algorithms to discover periodic patterns. Achieving this goal implies checking all possible periods to detect a cycle (i.e., this involves examining all possible periods from 1 to $\frac{n}{2}$, such as n is the sequence's length).

Algorithm	Length	S_{T_3}	
		Avg	SDev
Apriori	10	1,245	0,015
Apriori	50	13,446	0,918
Apriori	100	36,346	0,327
PPA	10	0,517	0,014
PPA	50	0,930	0,010
PPA	100	1,207	0,007
HashCycle	10	0,060	0,001
HasgCycle	50	0,146	0,001
HashCycle	100	0,236	0,002

Table 4: Experiments with data extracted from Twitter (S_{T_3}).

On the other hand, a few algorithms to look for cyclical patterns can be found in the literature. Moreover, some of them use mathematical techniques such as autocorrelation, which can not be used to compare with our algorithm. Instead, we have chosen the PPA algorithm, which has an excellent performance. Although PPA has been designed to find perfect and imperfect period patterns, it can find periodic cyclical patterns. Similarly, we have used the Apriori algorithm to have an upper bound since its performance has exponential behaviour. One should observe that both PPA and Apriori use minimum support, which is working to better compare the processing times among them

As already mentioned, a cyclical pattern is equivalent to a periodic pattern; however, the converse is not true. Furthermore, let's consider the following sequence $S = \{a\}\{f\}\{g\}\{x\}\{d\}\{c\}\{e\}\{c\}\{x\}\{c\}\{c\}\{e\}\{c\}\{h\}\{x\}$. If we consider $l = 5$, and $l_b = 2$, with $t_b = \{a\}$, then $\{\{x\}\{d\}, \{x\}\{c\}, \{h\}\{x\}\}$ are possible patterns, however, $\{x\}$ is always within $\{+\}\{+\}$ with $l_b = 2$ (i.e., $\{x\}$ can be in any of the two positions within the l_b). We denominated $\{+\}$, when an event such as $\{x\}$ occurs in any of all position with $\{+\}$. The latter provides new patterns that can occur within an interval, but only sometimes in the same position. For example, consider a postman who usually has to deliver parcels on an island, sometimes on Monday or Tuesday, depending on the weather. In this way, Monday and Tuesday do not correspond to a period from a periodic point of view; this is in contrast to a cyclical pattern, where both days conform to a recognizable pattern. In that regard, please note that in this paper, we are only concerned with proving the algorithms' efficiency when the cyclical patterns are periodic. Nonetheless, we believe that looking for cyclical patterns that there are not periodic is a good challenge since this type of problem is not common in Data Science.

On the other side, in this paper, we did not analyze the tweets' content but rather analyzed the time and place from where those are emitted. These cyclical patterns can indicate some interaction among users, suggesting something is happening around them. Furthermore, we believe that finding this kind of pattern makes it possible to analyze tweets' content better. In this way, spatio-temporal patterns can be exceptionally useful in social networks since these can prune tweets whose content is irrelevant. This latter can provide more efficient processing of tweets' content.

Regarding running times, the minimum support was instantiated in 1 in all experiments. Roughly, the running times decrease considerably when the minimum support increases, even for the Apriori algorithm. This latter is due to a significant pruning occurring every time the minimum support is raised. On the other hand, a successful search takes a maximum time $\Theta(1 + \alpha)$ when it achieves the list's end (according to *Theorem 2*). Accordingly, further experimentations should be conducted by varying the length of the interval (i.e. l_b) in which the patterns can occur.

7 Conclusion and Future Work

This paper introduces a novel and efficient algorithm named HashCycle, designed to identify cyclical patterns on Spatio-temporal data. HashCycle is characterized as a hash Table, where individual slots correspond to the events within a given sequence. Each slot contains linked lists that represent the relative positions of events, with these positions having a maximum length equal to the cycle period. Linked lists are obtained through collisions, ensuring that only those lists containing two or more nodes contribute to the formation of a valid cyclical pattern. Notably, HashCycle demonstrates exceptional space-saving capabilities by discarding lists that cannot form a complete cycle. Additionally, it is essential to highlight that HashCycle operates effectively without requiring a minimum support threshold, enabling the detection of all cyclical patterns appearing at least twice within a single execution. The time complexity for HashCycle implies $\Theta(1 + \alpha)$, while PPA takes $O(m^{m+1}n^2)$, and Apriori achieves $O(n\sqrt{m^n})$. The algorithms were evaluated over two Spatio-temporal datasets, one synthetic and another real, considering in this latter the location where tweets were emitted. Empirical results show that HashCycle is not only more efficient than Apriori and PPA but also scalable.

Ideas for future work include two threads. The first thread involves implementing algorithms such as MS-Apriori, FP-Growth, and Max-Subpattern; extending the experimental environments; increasing periods and sequence lengths; and considering different supports. The second thread involves varying the length of the interval within which the patterns can occur. The latter should yield new patterns that are discarded when the patterns are periodic.

Acknowledgements

This research was supported by Universidad del Bío-Bío, Chile, under Grants No. DIUBB GI 195212/EF, INN I+D 23-53, and DIUBB 2130253 IF/R.

References

- [Agrawal and Srikant, 1994] Agrawal, R. and Srikant, R. (1994). Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases, VLDB '94*, page 487–499, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- [Bhattacharya et al., 2022] Bhattacharya, S., Somayaji, S. R. K., Gadekallu, T. R., Alazab, M., and Maddikunta, P. K. R. (2022). A review on deep learning for future smart cities. *Internet Technology Letters*, 5(1):e187.

- [Bora et al., 2013] Bora, Zaytsev, Chang, and Maheswaran (2013). Gang networks, neighborhoods and holidays: Spatiotemporal patterns in social media. In *2013 International Conference on Social Computing (SocialCom)*, volume 0, pages 93–101.
- [Bunker et al., 2021] Bunker, R., Fujii, K., Hanada, H., and Takeuchi, I. (2021). Supervised sequential pattern mining of event sequences in sport to identify important patterns of play: An application to rugby union. *PLoS One*, 16(9):e0256329.
- [Cheng and Wicks, 2014] Cheng, T. and Wicks, T. (2014). Event detection using twitter: a spatio-temporal approach. *PLoS One*, 9(6):e97807.
- [Dambrowski et al., 2012] Dambrowski, J., Pichlmaier, S., and Jossen, A. (2012). Mathematical methods for classification of state-of-charge time series for cycle lifetime prediction. In *Advanced Automotive Battery Conference*, Europe.
- [Endo et al., 1974] Endo, T., Mitsunaga, K., Takahashi, K., Kobayashi, K., and Matsuishi, M. (1974). Damage evaluation of metals for random or varying loading—three aspects of rain flow method. *Mechanical Behavior of Materials*, (1):371–380.
- [Gaede and Günther, 1998] Gaede, V. and Günther, O. (1998). Multidimensional access methods. *ACM Comput. Surv.*, 30(2):170–231.
- [Gao et al., 2018] Gao, Y., Wang, S., Padmanabhan, A., Yin, J., and Cao, G. (2018). Mapping spatiotemporal patterns of events using social media: a case study of influenza trends. *Int. J. Geogr. Inf. Sci.*, 32(3):425–449.
- [Gupta, 2022] Gupta, M. (2022). Mathematically determining patterns in time series with codes. <http://bit.ly/490WJrU>.
- [Gutiérrez-Soto et al., 2022] Gutiérrez-Soto, C., Gutiérrez-Bunster, T., and Fuentes, G. (2022). A new and efficient algorithm to look for periodic patterns on spatio-temporal databases. *J. Intell. Fuzzy Syst.*, 42(5):4563–4572.
- [Han et al., 1999] Han, J., Dong, G., and Yin, Y. (1999). Efficient mining of partial periodic patterns in time series database. In *Proceedings 15th International Conference on Data Engineering*, pages 106–115.
- [Ishida, 2010] Ishida, K. (2010). Periodic topic mining from massive amounts of data. In *2010 International Conference on Technologies and Applications of Artificial Intelligence*, pages 379–386.
- [Jamshed et al., 2020] Jamshed, A., Mallick, B., and Kumar, P. (2020). Deep learning-based sequential pattern mining for progressive database. *Soft Computing*, 24(22):17233–17246.
- [Kandt and Batty, 2021] Kandt, J. and Batty, M. (2021). Smart cities, big data and urban policy: Towards urban analytics for the long run. *Cities*, 109:102992.
- [Koylu, 2019] Koylu, C. (2019). Modeling and visualizing semantic and spatio-temporal evolution of topics in interpersonal communication on twitter. *International Journal of Geographical Information Science*, 33(4):805–832.
- [Lahiri and Berger-Wolf, 2010] Lahiri, M. and Berger-Wolf, T. Y. (2010). Periodic subgraph mining in dynamic networks. *Knowl. Inf. Syst.*, 24(3):467–497.
- [Li et al., 2019] Li, X., Chen, W., Chan, C., Li, B., and Song, X. (2019). Multi-sensor fusion methodology for enhanced land vehicle positioning. *Inf. Fusion*, 46:51–62.
- [Li et al., 2011] Li, Z., Han, J., Ji, M., Tang, L.-A., Yu, Y., Ding, B., Lee, J.-G., and Kays, R. (2011). MoveMine: Mining moving object data for discovery of animal movement patterns. *ACM Trans. Intell. Syst. Technol.*, 2(4):1–32.
- [Lü et al., 2019] Lü, G., Batty, M., Strobl, J., Lin, H., Zhu, A.-X., and Chen, M. (2019). Reflections and speculations on the progress in geographic information systems (GIS): a geographic perspective. *Int. J. Geogr. Inf. Sci.*, 33(2):346–367.

- [Lv et al., 2013] Lv, M., Chen, L., and Chen, G. (2013). Mining user similarity based on routine activities. *Inf. Sci.*, 236:17–32.
- [Malode et al., 2015] Malode, Y. B., Khadse, D. B., and Jamthe, D. V. (2015). Efficient periodicity mining using circular autocorrelation in time series data. *International Research Journal of Engineering and Technology (IRJET)*, 03(3):430–436.
- [Molinaro and Orzes, 2022] Molinaro, M. and Orzes, G. (2022). From forest to finished products: The contribution of industry 4.0 technologies to the wood sector. *Computers in Industry*, 138:103637.
- [Nakata and Takeuchi, 2004] Nakata, T. and Takeuchi, J.-I. (2004). Mining traffic data from probe-car system for travel time prediction. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '04*, pages 817–822, New York, NY, USA. Association for Computing Machinery.
- [Ozden et al., 1998] Ozden, B., Ramaswamy, S., and Silberschatz, A. (1998). Cyclic association rules. In *Proceedings 14th International Conference on Data Engineering*, pages 412–421.
- [Parthasarathy et al., 2006] Parthasarathy, S., Mehta, S., and Srinivasan, S. (2006). Robust periodicity detection algorithms. In *Proceedings of the 15th ACM international conference on Information and knowledge management, CIKM '06*, pages 874–875, New York, NY, USA. Association for Computing Machinery.
- [Pillai et al., 2013] Pillai, K. G., Angryk, R. A., and Aydin, B. (2013). A filter-and-refine approach to mine spatiotemporal co-occurrences. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL '13*, pages 104–113, New York, NY, USA. Association for Computing Machinery.
- [Pillai et al., 2012] Pillai, K. G., Angryk, R. A., Banda, J. M., Schuh, M. A., and Wylie, T. (2012). Spatio-temporal co-occurrence pattern mining in data sets with evolving regions. In *Proceedings of the 2012 IEEE 12th International Conference on Data Mining Workshops, ICDMW '12*, pages 805–812, USA. IEEE Computer Society.
- [Rejeb et al., 2022] Rejeb, A., Suhaiza, Z., Rejeb, K., Seuring, S., and Treiblmaier, H. (2022). The internet of things and the circular economy: A systematic literature review and research agenda. *Journal of Cleaner Production*, 350:131439.
- [Revesz, 2009] Revesz, P. (2009). *Spatiotemporal Interpolation Algorithms*, pages 2736–2739. Springer US, Boston, MA.
- [Saneep and Swapnil, 2015] Saneep, K. and Swapnil, K. (2015). Design & implementation of efficient periodicity mining technique for time series data. *International Journal of Advanced Research in Computer and Communication Engineering*, 4(5):439–444.
- [Song et al., 2015] Song, Y., Hu, Z., Leng, X., Tian, H., Yang, K., and Ke, X. (2015). Friendship influence on mobile behavior of location based social network users. *Journal of Communications and Networks*, 17(2):126–132.
- [Tirumalasetty et al., 2015] Tirumalasetty, S., Jadda, A., and Edara, S. R. (2015). An enhanced apriori algorithm for discovering frequent patterns with optimal number of scans.
- [Yang et al., 2013] Yang, K.-J., Hong, T.-P., Chen, Y.-M., and Lan, G.-C. (2013). Projection-based partial periodic pattern mining for event sequences. *Expert Syst. Appl.*, 40(10):4232–4240.