big data and cognitive computing

MDPI

Article

An Efficient Probabilistic Algorithm to Detect Periodic Patterns in Spatio-Temporal Datasets

Claudio Gutiérrez-Soto [1], Patricio Galdames [2] and Marco A. Palomino [3,*]

[1] Departamento de Sistemas de Información, Universidad del Bío-Bío, Concepción 4030000, Chile; cogutier@ubiobio.cl
[2] Facultad de Ingeniería, Arquitectura y Diseño, Universidad San Sebastián, Concepción 4090000, Chile; patricio.galdames@uss.cl
[3] School of Natural and Computing Sciences, University of Aberdeen, Aberdeen AB24 3UE, UK
* Correspondence: marco.palomino@abdn.ac.uk

Abstract: Deriving insight from data is a challenging task for researchers and practitioners, especially when working on spatio-temporal domains. If pattern searching is involved, the complications introduced by temporal data dimensions create additional obstacles, as traditional data mining techniques are insufficient to address *spatio-temporal databases* (STDBs). We hereby present a new algorithm, which we refer to as *F1/FP*, and can be described as a probabilistic version of the *Minus-F1* algorithm to look for periodic patterns. To the best of our knowledge, no previous work has compared the most cited algorithms in the literature to look for periodic patterns—namely, *Apriori*, *MS-Apriori*, *FP-Growth*, *Max-Subpattern*, and *PPA*. Thus, we have carried out such comparisons and then evaluated our algorithm empirically using two datasets, showcasing its ability to handle different types of periodicity and data distributions. By conducting such a comprehensive comparative analysis, we have demonstrated that our newly proposed algorithm has a smaller complexity than the existing alternatives and speeds up the performance regardless of the size of the dataset. We expect our work to contribute greatly to the mining of astronomical data and the permanently growing online streams derived from social media.

Keywords: sequential pattern mining; spatio-temporal databases; frequent item sets

Citation: Gutiérrez-Soto, C.; Galdames, P.; Palomino, M.A. An Efficient Probabilistic Algorithm to Detect Periodic Patterns in Spatio-Temporal Datasets. *Big Data Cogn. Comput.* **2024**, *8*, 59. https://doi.org/10.3390/bdcc8060059

Academic Editors: Luca Virgili, Francesco Cauteruccio and Enrico Corradini

Received: 18 April 2024
Revised: 20 May 2024
Accepted: 29 May 2024
Published: 3 June 2024

1. Introduction

Recent technological developments have led to a *data deluge* [1], a scenario where more data are generated than can be successfully and efficiently managed or capped. This results in missed chances to analyze and interpret data to make informed decisions.

When decision making calls for pattern discovery, the complexity is further expanded if the data have spatio-temporal features, because traditional algorithms are not meant to handle the search for correlations which have a time dimension. This is, for example, the case of global positioning systems [2] and geographic information systems [3], which can be represented as *spatio-temporal databases* (STDBs)—that is, extensions to existing information systems that include time to better describe a dynamic environment [4].

The exploitation of STDBs can provide valuable knowledge, for instance, in the context of road traffic control and monitoring [5], weather analysis [6], and location-based sociological behavior in social networks [7]. However, as stated above, traditional data mining techniques cannot be directly applied to STDBs, which complicates not only data exploitation but also processing times.

We are interested in the discovery of *periodic patterns*, which can be seen as events occurring with a certain "periodicity"—for example, the subway's arrival at Central Park Station every 15 min defines a periodic pattern. A *period* corresponds to any unit of time, such as hours, days, weeks, et cetera. To be precise, a period is the time elapsed between two occurrences of a pattern, and it can be counted in terms of time or a number of transactions.

Sequential pattern mining is also concerned with finding statistically relevant patterns where data appear in a sequence [8]. The sequence is analyzed in such a manner that the possible patterns satisfy a minimum threshold while considering the length of the periods to be analyzed. From the point of view of performance, the discovery of valuable knowledge depends on two aspects: the volume of data and the processing power. Hence, in a context where data grow exponentially, it is critical to ensure the use of efficient algorithms, regardless of the available processing power.

Problem Definition

Let $o(s', t)$ be a spatio-temporal object defined by a point in time t and a spatial location s'. A change in the shape of the object or in the object's location is known as an *event*. We will denote an event as $e(o_x, t_i)$, where o_x is the object at a location s'_m and a time t_i. For simplicity, the space where the objects are located is segmented into a set of $n \times n$ disjoint cells with equal sizes. A cell is denoted as s'_m, and a sequence of localized events for the object o_x is denoted as S_x. Events belonging to S_x take place over a time series τ, such as $\tau = \{t_1, \ldots, t_n\}$, where $t_i < t_{i+1}$.

Definition 1. *Given a minimum support $sup(X)_{min}$ provided by a user, then $sup(X)$ is a p-periodic pattern if and only if $sup(X) \geq sup(X)_{min}$, such that the length of X is p and p corresponds to the period. X is a p-periodic pattern over S_x if it satisfies two user requirements: p and $sup(X)_{min}$. To illustrate this, consider the sequence:*

$$S_x = \{\{a\}\{b\}\{c\}\{a\}\{d\}\{c\}\{a\}\{e\}\{c\}\},$$

where $sup(X)_{min} = 1/3$ and $p = 3$. There are three subsequences, each containing three events. Thus, it is feasible to obtain the p-periodic pattern $\{a\}\{*\}\{c\}$, which corresponds to a partial periodic pattern, because $\{*\}$ can represent any event. This pattern is also a perfect periodic pattern, as it appears across all three subsequences.

The main contributions of this paper are as follows :

Extensive experimentation: To the best of our knowledge, no previous work has compared, empirically, the performance of the most cited algorithms based on association rules, such as *Apriori* [9], *MS-Apriori* [10], *FP-Growth* [11], *PPA* [12], and *Max-Subpattern* [11]. Thus, we have conducted a comprehensive comparison of these algorithms over two STDBs—first, a synthetic one, then a real one. As part of our experiments, we have also included the *Minus-F1* algorithm [13], which has been proven to achieve good results, and a new probabilistic version of it, which we have developed.

An efficient probabilistic algorithm: Although recent developments have produced several off-the-shelf libraries for pattern mining—for instance, `apyori` [14] is a library that implements the Apriori algorithm in *Python*—our experiments have confirmed that the performance of the most well-known algorithms is not ideal for STDBs. Thus, we have developed a new, probabilistic version of the *Minus-F1* algorithm [13], which we refer to as *F1/FP*. This new algorithm allows for periodic pattern discovery in STDBs. As in the case of Minus-F1, F1/FP is an algorithm of *Las Vegas* type [15], which always provides the correct answer when searching for a pattern, and has a polynomial behavior matched with a better performance in STDBs.

Complexity analysis: A calculation of the complexity of the F1/FP algorithm. The complexities of association rule algorithms have not been discussed sufficiently in the literature. Indeed, we have struggled to find sources where this kind of analysis is undertaken. Thus, we have endeavoured to prove that the complexity of our newly proposed algorithm is better than that of the alternatives.

We expect our work to contribute significantly towards future research on pattern searching, especially in the case of the exploration of massive datasets—such as those

required for the mining of astronomical data—and online streams which continue to grow uninterruptedly—such as those derived from social media.

The remainder of this paper is organized as follows: Section 2 consists of a bibliographical review of pattern searching. Section 3 presents the main algorithms based on association rules, and Section 4 analyzes the complexity of our proposal. Section 5 reports on the experimental environment and Section 6 introduces our results. Lastly, Section 7 offers our conclusions and comments on future work.

2. Related Work

There are three types of sequential pattern-mining algorithms: machine learning algorithms, algorithms based on mathematical techniques, and algorithms based on association rules. Machine learning algorithms require an objective function and a training dataset to define "correct" patterns [16,17]. This approach often involves a complex model selection process and hyperparameter tuning, which can be challenging for users who lack sufficient domain knowledge and experience. Thus, this approach is unsuitable for users who are not well versed in the intricacies of training and tuning machine learning models.

Algorithms based on mathematical techniques involve the utilization of the Fourier transform to calculate the circular autocorrelation [18]. This allows customization. For instance, Khanna and Kasurkar [19] addressed three types of periodicity—symbol periodicity, segment periodicity, and partial periodicity—by proposing corresponding variants of an algorithm based on autocorrelation. Methods based on mathematics are also robust against noise and efficient at extracting partial periodic patterns, without additional domain knowledge. Regrettably, they prioritize computational efficiency by employing approximations, which may miss some periodic patterns [13]. In other words, mathematical methods trade off the guarantee of finding all the qualifying patterns for faster execution times.

Association rule mining algorithms are those derived from the Apriori-based association rule proposed by Agrawal and Srikant [9]. These algorithms exploit the fact that "any superset of an infrequent item set is also infrequent". Indeed, Apriori identifies frequent item sets from smaller to larger candidates by pruning infrequent ones to prevent an explosion of the number of combinations to be examined.

Even though Apriori remains a well-regarded algorithm [20], it has limitations. First, it only allows for a single *minimum support* (MS), which can restrict its scope. Second, its efficiency may be lacking in certain situations. To address the first drawback, the MS-Apriori algorithm [10] has been developed to enable the discovery of frequent patterns across multiple thresholds. To address the second drawback, optimization strategies have been used to take advantage of the inherent properties of periodic pattern mining [21,22]. For example, it is not necessary to assess the frequency of an item set in position t if it is not frequent at any position contained within the cycles involving t. Also, other researchers have looked into algorithms that use properties specific to the types of patterns they are interested in, for instance, partial periodic patterns [23], asynchronous periodic patterns [24], symbol periodicity, sequence periodicity, and segment periodicity [25].

Spatio-temporal databases are another area which extends the scope of the problem with many new applications, such as disease diffusion analysis [26], user activity analysis [27], and local trend discovery in social networks [28,29]. Several approaches have been proposed to deal with spatial information [30], treating it as a continuous variable [31,32], formulating it as a dynamic graph mining problem [33], and encoding spatial features as discrete symbols [13]. We have adopted the discrete symbol encoding approach to fully exploit our former research on sequential periodic pattern mining [13].

Han et al. [11] proposed the Max-Subpattern Hit-Set algorithm, often referred to simply as Max-Subpattern. They based their development on a custom data structure called a *max-subpattern tree* to efficiently generate larger partial periodic patterns from combinations of smaller patterns. Yang et al. [12] proposed the projection-based partial periodic pattern algorithm (PPA), derived from a strategy to encode events in tuples. The empirical results show that the PPA algorithm is better at discovering partial periodic

patterns than Max-Subpattern and Apriori. Han et al. [34] also proposed another algorithm called *partial frequent pattern growth* (PFP-Growth).

PFP-Growth has two stages: the first stage constructs an FP-tree, and the second stage recursively projects the tree to output a complete set of frequent patterns. Experiments were carried out comparing PFP-Growth with the Max-Subpattern algorithm on synthetic data. Results show that PFP-Growth performs better than Max-Subpattern.

Then, Gutiérrez-Soto et al. suggested the Minus-F1 algorithm in 2022 [13]. This is an algorithm designed specifically to search for periodic patterns in STDBs. Gutiérrez-Soto et al. showed that Minus-F1 has a polynomial behavior, which makes it more efficient than other alternatives, such as Apriori, Max-Subpattern, and the PPA. Recently, Gutiérrez-Soto et al. [35] proposed an alternative called *HashCycle* to find cyclical patterns. Although highly relevant, HashCycle is not appropriate for periodic pattern discovery.

Xun et al. [36] proposed a new pattern called a *relevant partial periodic pattern* and its corresponding mining algorithm (*PMMS-Eclat*) to effectively reflect and mine the correlations of multi-source time series data. PMMS-Eclat uses an improved version of Eclat to determine frequent partial periodic patterns and then applies the locality-sensitive hashing (LSH) principle to capture the correlation among these patterns [37].

Jiang et al. [38] addressed the discovery of periodic frequent travel patterns of individual metro passengers considering different time granularities and station attributes. The authors proposed a new pattern called a "periodic frequent passenger traffic pattern with time granularities and station attributes" (PFPTS) and developed a complete mining algorithm with a PFPTS-Tree structure. The proposed algorithm was evaluated on real smart card data collected by an automatic fare collection system in a large metro network. As opposed to Jiang et al., our work can be applied in different situations rather than specifically on individual travellers contexts.

Whilst existing algorithms have been designed to handle various aspects of periodic pattern mining and spatio-temporal data, they often focus on optimizing computational efficiency or addressing specific pattern types. In contrast, our work presents a novel probabilistic variant of the Minus-F1 algorithm that aims to balance efficiency and effectiveness in a wide range of scenarios. The proposed algorithm is exhaustively evaluated against most of the previously mentioned algorithms using two datasets with diverse characteristics, showcasing its ability to handle different types of periodicity and data distributions. By conducting a comprehensive comparative analysis, we will highlight the unique contributions and advantages of our probabilistic variant of Minus-F1.

3. Algorithms

Sequential pattern mining is concerned with finding statistically relevant data patterns where the values appear in a sequence [8]. Several algorithms have been designed for this purpose, and we want to compare our newly suggested alternative with the most well-regarded options, namely, Apriori, Max-Subpattern, PPA, Minus-F1, and FP-Growth. We will describe these options below and illustrate our explanations with examples.

3.1. Apriori

Apriori is an algorithm for frequent item mining on relational databases [9]. It identifies items retrieved frequently in a database and creates a set containing such items. Over time, the set becomes larger, as items continue to be added if they are retrieved often. These sets can later be used to establish association rules [39], which highlight trends in the database. Although Apriori is not originally designed to have a temporal dimension, we have amended it to include it.

Consider the following example. Let us assume that the string below represents a time series with periodicity four—the periodicity has been determined in advance. Note that each character in the string represents a separate event, and the events within curly braces are those that occur simultaneously.

$$a\{b,c\}ddab\{c,d\}daabbacbda\{b,d\}da$$

Given that the periodicity of the time series is four, we can confirm that the number of periods is five. We have used hyphens to separate each period in the line below.

$$a\{b,c\}dd - ab\{c,d\}d - aabb - acbd - a\{b,d\}da$$

Apriori identifies the sets of frequent items by making subsequent passes through the database. In the first pass, it gathers the set of frequent items of size 1; then, in the second pass, the set of frequent items of size 2 and so on.

Let us call F_k the set of frequent items of size k. Then, assuming a minimum support of 3, F_1 can be derived from the following candidates:

$$\begin{aligned} F_{1,Candidates}: \quad & \{a***:\tfrac{5}{5}, *a**:\tfrac{1}{5}, **a*:\tfrac{1}{5}, ***a:\tfrac{1}{5} \\ & *b**:\tfrac{3}{5}, **b*:\tfrac{2}{5}, ***b:\tfrac{1}{5} \\ & *c**:\tfrac{2}{5}, **c*:\tfrac{1}{5} \\ & *d**:\tfrac{1}{5}, **d*:\tfrac{3}{5}, ***d:\tfrac{3}{5}\} \end{aligned}$$

Thus,

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

Subsequently, F_2 can be derived from the following candidates,

$$\begin{aligned} F_{2,Candidates}: \quad & \{ab**:\tfrac{3}{5}, ac**:\tfrac{2}{5}, a*d*:\tfrac{3}{5}, a**d:\tfrac{3}{5}, \\ & *bd*:\tfrac{3}{5}, *b*d:\tfrac{2}{5}, *c*d:\tfrac{2}{5}, **dd:\tfrac{2}{5}\} \end{aligned}$$

Therefore,

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

Finally, there is only one candidate for F_3,

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

The algorithm finishes when $F_K = \emptyset$. Then, we finish with F_3 in this example, as the number of events in F_3 cannot generate an F_4 set.

3.2. Max-Subpattern

Max-Subpattern was originally proposed by Han et al. [11] as an attempt to reduce the number of sets to determine periodic patterns [40]. It builds as many trees as the number of periods we encounter in a time series, representing a sequence of events. However, period 1, which is equivalent to a period formed by a single event, is not taken into account. If a sequence has size n, the maximum number of periods to evaluate is $\frac{n}{2}$. Thus, Max-Subpattern builds up to $\frac{n}{2} - 1$ trees.

Let us call C_{max} the root of the tree. Then, for each set of candidates $F_{k,Candidates}$, there is a different C_{max}. Also, each level of the tree will have subpatterns. For instance, if C_{max} is formed by four events, the next level in the tree (Level 1) will be formed by four nodes, and each node will represent a subpattern composed of $|C_{max}| - 1$ events. Then, Level 2 is formed by nodes with $|C_{max}| - 2$ events whose ancestor belongs to Level 1. Each node is made up of at least two events, that is, without considering F_1. Thus, the maximum height for each tree is $|C_{max}| - 1$.

Let us consider the same example used for Apriori in Section 3.1. Once F_1 has been determined, C_{max} is formed. Hence,

$$C_{max} = abdd$$

Then, we proceed to find subpattern hits, discarding all the matches with only one non-* element .

$$H_{Candidates} : \quad \{abdd, abdd, a**, a**d, abd*\}$$

Thus, H is formed by:

$$H = \{abdd : 2, a**d : 1, abd* : 1\}$$

3.3. PPA

After discovering that Max-Subpattern spends a large amount of time calculating frequency counts from redundant candidate nodes, Yang et al. [12] developed the projection-based partial periodic patterns algorithm—abbreviated as PPA—for mining partial periodic patterns with a specific period length in an event sequence.

The PPA starts by going over the time series which represents the sequence of events and splits it into partial periods of size l. Afterwards, each event is codified—that is, the position of each event inside the partial period is recorded. Codified events can be seen as a matrix, where the first row corresponds to the first codified events and each column corresponds to the event's position inside the partial periods. The matrix was referred to by Yang et al. as an *encoded period segment database* (EPSD) [12].

By following this approach, it is possible to count the instances of each event by column, and the result is used to check whether the events comply with the required support. Consider Apriori's example defined in Section 3.1. Specifically, consider a particular instance of the original example for Apriori, namely,

$$abdd \; - \; abdd - aabb - acbd - abda$$

Consequently, the matrix is defined as follows,

$$EPSD = \begin{pmatrix} a_1 & b_2 & d_3 & d_4 \\ a_1 & b_2 & d_3 & d_4 \\ a_1 & a_2 & b_3 & b_4 \\ a_1 & c_2 & b_3 & d_4 \\ a_1 & b_2 & d_3 & a_4 \end{pmatrix}$$

where the element x_i corresponds to event x in position i. Once the instances of each event are counted by column, and the minimum support is satisfied, a candidate subsequence can be derived. Then, the events that form this subsequence are sorted, considering first the partial positions and then the lexicographic nomenclature of each event. The last subsequence S_c is equivalent to F_1. Indeed, according to Yang et al. [12], S_c is used to look for the other $F_{k,Candidates}$ patterns. Each event of S_c is used as a prefix to obtain the patterns that comply with the minimum support over the EPSD. Finally, all the F_k sets that fulfil the minimum support are gathered.

3.4. Minus-F1

Minus-F1 operates by using two counters: one which is increased by 1 every time there is a match with the candidate pattern, and a second one which decreases until it reaches zero when the subsequence is consumed. In the first run of the algorithm, the sequence's probability distribution is calculated—this can be seen as capturing the entropy of all the events in the sequence. To achieve this, Minus-F1 finds out how many times each event occurs. When an event occurs, its counter is decreased. Thus, when the counter reaches zero, we can confirm that it is unnecessary to keep looking for it—it can no longer occur.

The worst-case scenario for Minus-F1 happens when the events are distributed uniformly [13]. In contrast, when the distribution is not uniform, the algorithm performs the pruning efficiently. To illustrate this, let us consider the following sequence S, which comprises the subsequences $s_1 = abc$, $s_2 = abj$, $s_3 = efg$, and $s_4 = hij$, namely,

$$S = \{abc - abj - efg - hij\}.$$

Note that all the subsequences have period 3. Assuming a minimum support of 2, the only two subsequences $S_{i,j}$ which satisfy the minimum support and form a partial pattern are $S_{1,1} = a$ and $S_{1,2} = b$. In other words, $ab*$ is the only partial pattern.

Once the subsequences s_1 and s_2 have been consumed, it makes no sense to continue searching for them—in our example, the events a and b cannot occur in s_3 and s_4. Hence, we can prune the search space.

It appears that Minus-F1 is mainly affected by the size of the period [13], as opposed to the number of patterns found, which is different to the rest of the algorithms reviewed here. In fact, Minus-F1 goes through the entire sequence of events once for each period under consideration. Thus, Gutiérrez-Soto et al. [13] have pointed out that Minus-F1's best performance is achieved as the minimum support tends to zero. We aimed to fix this issue in the new algorithm that we are proposing.

3.5. FP-Growth

FP-Growth was designed to derive sets of frequent items from sequences without a pre-defined period. The algorithm begins by creating a table comprising the frequent items which satisfy the minimum support. Then, the table is sorted in descending order.

Let us say that the items which satisfy the minimum support are as follows:

$$caba - cabd - cabc - cabb$$

Then, FP-Growth removes from the items the segments that do not satisfy the minimum support and separates them to search for partial patterns. Finally, the patterns are sorted according to the position they have in the original segments. In the case of our example, the results are displayed in Table 1:

Table 1. Frequent patterns .

Frequent Pattern	Order	Support
C	c***	100%
A	*a**	100%
B	**b*	100%
bca	cab*	100%
bc	c*b*	100%
ba	*ab*	100%
ac	ca**	100%

3.6. Minus-F1's (Probabilistic Version)

Our version of Minus-F1, which we have called F1/FP, is a Las Vegas type of algorithm, which always provides the correct answer. This means that its performance in the worst-case scenario corresponds to the deterministic algorithm's performance. Note that this situation arises only when the probability distribution of the algorithm's input data reaches the worst case. Although this is uncommon, it depends on the probability distribution. Therefore, the time complexities for this type of algorithm are expressed as expected time, denoted by $\Theta(f(n))$.

F1/FP operates similarly to Minus-F1, except that, when searching for subsequences, these are selected randomly, assuming their occurrence likelihood follows a uniform distribution. This can be seen in Line 9—the swap procedure—of Algorithm 1, where we have listed the pseudo-code for F1/FP to illustrate our explanation. This simple modification of Minus-F1 provides a better performance. It is worth noting that the literature has plenty of such subtle improvements, which result in better performances and running times.

Algorithm 1 Probabilistic Minus-F1

Require: $support \geq 0 \wedge S \neq \emptyset \wedge n \geq 0 \wedge m \geq 0 \wedge Table[], F_1$
Ensure: $F' = \cup F_{i=1}^{n/2}$
1: $F' = F_1$
2: **for** $i \leftarrow 1, i \leq m$ **do**
3: **for** $j \leftarrow 1, j \leq \frac{n}{2}$ **do**
4: $Table[i,j].Cont \leftarrow 0$
5: **end for**
6: **end for**
7: **for** $p \leftarrow 2, p \leq \frac{n}{2}$ **do**
8: **for** $i \leftarrow 1, i \leq \frac{n}{p}$ **do**
9: $s_{i'} = swap(s_i, s_{i+1} \ldots s_{\frac{n}{p}})$
10: **for** $j \leftarrow 1, j \leq p$ **do**
11: $m' \leftarrow KeyFunction(s_{i',j})$
12: $Table[m', j].Cont++$
13: $j \leftarrow j + 1$
14: **end for**
15: **end for**
16: **for** $i \leftarrow 1, i \leq \frac{n}{p}$ **do**
17: **for** $j \leftarrow 1, j \leq p$ **do**
18: $m' \leftarrow KeyFunction(s_{i',j})$
19: **if** $Table[m', j].Cont \geq support$ **then**
20: $F_p = F_p \cup Table[m', j]$
21: $Table[m', j].Cont--$
22: **end if**
23: **end for**
24: **end for**
25: $F' = F' \cup F_p$
26: **end for**

4. Time Complexity

To show how random swaps affect the running time, determined by its expected value, we provide the following definitions:

Definition 2. *Let $F(s)$ be a function determining the occurrence of subsequence s_i within the sequence S, such that:*

$$F(s_i) = \begin{cases} 1 & \text{if } s \text{ occurs at moment } t_i \text{ over } S. \\ 0 & \text{if } s \text{ does not occur at moment } t_i \text{ over } S. \end{cases}$$

Definition 3. *Let $Pr[s_i]$ be the probability of choosing some subsequence within sequence S, such that its position can be between $i+1$ and n', where n' is the number of sequences to carry out a swap ($n' = n/p$).*

We assume that all subsequences have the same probability of being selected—in other words, a uniform distribution is assumed. Thus, $Pr[s_i]$ is defined as follows:

$$Pr[s_i] = \frac{n' - i}{n'}$$

Definition 4. *Given a random subsequence $\mathfrak{s}$, whose position within S is i, the expected value to carry out a swap is defined as :*

$$E[\mathfrak{s}_i] = \sum_{i=i+1}^{n} F(s_i')Pr[s_i']$$

Lemma 1. *The number of swaps carried out by the probabilistic version of Minus-F1 is given by the number of subsequences* $n' - 1$.

Therefore, the time complexity to carry out a swap is given by:

$$E[S] \leq \Theta(n')$$

Proof by Induction: Base case (when $n' = 2$): Using the loop invariant in Lines 2–3 in Algorithm 1, we notice that there is a swap, which is equal to $n' - 1$. Note that S has two events. Then, the random event is chosen from the first event of the sequence. Thus,

$$E[S] = F(s'_2)Pr[s'_2] = 1\frac{1}{2} = 1\frac{1}{n'}$$

Therefore, $E[S] \leq \Theta(n')$.

Inductive steps: This case is provided when $n' \geq 3$, for any k-iteration from $i = 2$ until n', such that $k \leq n'$. Using the loop invariant in lines 2–3, there are always $n' - 1$ swaps. Without loss of generality, it is expressed as

$$\begin{aligned}
E|S] \leq & \\
& F(s'_2)Pr[e'_2]+ \\
& \ldots F(s'_k)Pr[s'_k] + F(s'_n)Pr[s'_n] \\
& \leq 1\frac{n'-2}{n'} + \ldots + 1\frac{n'-k}{n'} + 1\frac{n'-n'}{n'} \text{(by Definitions 2 and 3)} \\
& \leq 1\binom{0}{n'} + \ldots + k\binom{1}{n'} + (n'-1)\binom{1}{n'} \\
& \leq \frac{\binom{n'+1}{2}}{n'} \\
& \leq \Theta(n')
\end{aligned}$$

Although our random procedure provides notable improvements in running times, its time complexity does not change in general. This is because the sequence's length is n, and it must run through all subsequences of length $\frac{n}{p}$ over p periods. Consequently, running all subsequences s_i takes $\Theta(n)$. Without loss of generality, and given that all algorithm loops operate on subsequences chosen randomly, a portion of this version can be denoted as $\Theta(f(n))$, instead of using $O(f(n))$, except for the loops between lines 2 and 8—note that these loops are related to m events. Thus, since Minus-F1's time complexity is $O(mn^2)$; this probabilistic version can be characterized as $\Theta(mn^2)$, which is bounded by $O(mn^2)$. □

5. Experimentation

To check the algorithms' performance, two datasets were used. The first one is composed of synthetic data, and it was used to corroborate that each algorithm was implemented correctly—that is, to confirm that each algorithm was able to find the required patterns. Once correctness had been verified, we used a second dataset to confirm that the algorithms could handle real data. The second dataset is a sample of the Geolife GPS trajectory dataset [41].

Geolife records a broad range of users' outdoor movements, including daily routines—going to work or returning home—and activities like travelling to entertainment, shopping, and sport activities [41]. Geolife has been widely used in mobility pattern mining and location-based social networks [26], which are potential applications for our work. Therefore, we thought this dataset would fit our experimentation adequately.

The Geolife dataset comprises GPS trajectories undertaken by 182 people over a period of three years—between April 2007 and August 2012—and it was collected by *Microsoft Research Asia*. Each GPS trajectory is represented by a sequence of time-stamped points labelled by latitude, longitude, and altitude.

To characterize Geolife as an STDB for our experiments, the space was represented by a set of cells forming a grid. The location of each object within the grid was determined by its latitude and longitude. Time was modelled as a timestamp. At timestamp 0, all the objects are situated in their initial positions. Subsequently, objects move to different positions across the grid. An object's motion was characterized as a contiguous sequence of characters, facilitating pattern searching within the sequence.

It should be observed that our representation of motion can have an impact on pattern detection only if movement occurs within a time window whose granularity is smaller than what has been represented. For instance, if we were measuring time in minutes, we could lose some patterns occurring within seconds. However, this is not the case. The efficiency of the algorithms considered here does not depend on the granularity of the grid, but on the length of the sequence.

The results displayed below correspond to the average of five executions for each experiment. From an empirical perspective, the performance of each algorithm is determined by its running time. To define a pattern, a range of 2 to $\frac{n}{2}$ events was considered, where n represents the length of the sequence. This implies that all patterns consist of at least two events—at least one event repetition—occurring up to half the length of the sequence. For a pattern to be valid, it must occur at least twice within the sequence.

All experiments were limited to a maximum of 3 h—results exceeding this length are not shown. The experiments were carried out on a server equipped with an Intel Xeon Processor E3-1220 at 3.00 GHz and 16 GB of RAM operating at 2133 MHz with a 1 TB 7200 RPM hard drive, and running under Linux (Debian Jessie 8.4). Table 2 displays the abbreviations used later in our results to refer to the different algorithms.

Table 2. Algorithm abbreviations.

Acronym	Meaning
APR	Apriori algorithm
MSA	MS-Apriori algorithm
M-SP	Max-Subpattern algorithm
PPA	PPA algorithm
FP-G	FP-Growth algorithm
F1/FP	Our probabilistic version of Minus-F1
F1	Minus-F1 algorithm

5.1. Results Derived from the Synthetic Dataset

The experiments contemplated sequences of size 500, 750 and 1000, considering periods of 4, 8, 16, and 20. To link the running times with the corresponding computational complexities for each algorithm, two experiments were performed. The experiments cover pattern searching over the synthetic database which has a full pattern with period 48 and is repeated until achieving the sequence size. Supports for 25% (Tables 3–5), 50% (Tables 6–8), and 75% (Tables 9–11) were considered.

Table 3. Processing time (ms) for each algorithm using 500 sequences with a minimum support of 25% over synthetic data.

Algorithm		APR	M-SP	MSA	PPA	FP-G	F1/FP	F1
Period	4	42	483	11	12	7	5	6
	8	155	680	36	32	8	5	13
	12	1.9×10^4	932	3.4×10^4	3.2×10^4	3.3×10^3	6	23
	16	2.9×10^7	1.3×10^3	2.3×10^7	9.4×10^6	1.9×10^6	6	38
	20	5.8×10^7	1.9×10^3	180	1.0×10^6	11	10	53

Table 4. Processing time (ms) for each algorithm using 750 sequences with a minimum support of 25% over synthetic data.

Algorithm		APR	M-SP	MSA	PPA	FP-G	F1/FP	F1
Period	4	50	1.3×10^3	16	16	8	6	8
	8	184	1.6×10^3	52	32	9	6	18
	12	2.1×10^4	2.1×10^3	3.5×10^4	4.9×10^4	3.2×10^3	6	32
	16	3.0×10^7	2.9×10^3	2.3×10^7	1.4×10^7	1.8×10^6	8	51
	20	5.1×10^7	3.8×10^3	239	1.5×10^6	13	12	77

Table 5. Processing time (ms) for each algorithm using 1000 sequences with a minimum support of 25% over synthetic data.

Algorithm		APR	M-SP	MSA	PPA	FP-G	F1/FP	F1
Period	4	86	2.9×10^3	21	16	11	6	10
	8	220	3.0×10^3	68	57	11	7	22
	12	2.1×10^4	4.1×10^3	3.7×10^4	6.6×10^4	3.2×10^3	8	41
	16	2.7×10^7	5.2×10^3	2.3×10^7	1.9×10^7	1.8×10^6	9	67
	20	5.8×10^7	6.7×10^3	305	2.1×10^5	16	14	98

Table 6. Processing time (ms) for each algorithm using 500 sequences with a minimum support of 50% over synthetic data.

Algorithm		APR	M-SP	MSA	PPA	FP-G	F1/FP	F1
Period	4	13	435	2	3	6	3	5
	8	62	408	3	3	5	4	10
	12	1.5×10^3	464	1.0×10^4	8.8×10^3	1.6×10^3	5	18
	16	1.6×10^3	446	49	32	6	6	28
	20	1.7×10^3	542	6	6	7	6	42

Table 7. Processing time (ms) for each algorithm using 750 sequences with a minimum support of 50% over synthetic data.

Algorithm		APR	M-SP	MSA	PPA	FP-G	F1/FP	F1
Period	4	13	1.2×10^3	3	3	7	4	6
	8	73	1.2×10^3	4	3	7	4	18
	12	1.8×10^3	1.3×10^3	1.1×10^4	1.3×10^4	1.6×10^3	6	24
	16	1.9×10^3	1.2×10^3	70	45	8	8	38
	20	2.1×10^3	1.3×10^3	7	6	9	8	60

Table 8. Processing Time (ms) for each algorithm using 1000 sequences with a minimum support of 50% over synthetic data.

Algorithm		APR	M-SP	MSA	PPA	FP-G	F1/FP	F1
Period	4	16	2.6×10^3	4	4	10	4	7
	8	81	2.8×10^3	5	5	8	5	17
	12	2.1×10^3	2.8×10^3	1.2×10^4	1.7×10^4	1.6×10^3	7	30
	16	2.2×10^3	2.9×10^3	93	60	9	8	49
	20	2.5×10^3	3.0×10^3	9	8	10	9	78

Table 9. Processing time (ms) for each algorithm using 500 sequences with a minimum support of 75% over synthetic data.

Algorithm		APR	M-SP	MSA	PPA	FP-G	F1/FP	F1
Period	4	13	434	2	3	5	2	5
	8	25	440	3	3	6	3	10
	12	38	434	4	3	6	3	17
	16	58	449	5	4	6	5	26
	20	74	454	6	6	7	6	38

Table 10. Processing time (ms) for each algorithm using 750 sequences with a minimum support of 75% over synthetic data.

Algorithm		APR	M-SP	MSA	PPA	FP-G	F1/FP	F1
Period	4	14	1.2×10^3	4	3	7	4	6
	8	32	1.2×10^3	5	4	8	4	13
	12	43	1.2×10^3	5	4	6	4	23
	16	59	1.2×10^3	7	5	6	5	35
	20	86	1.3×10^3	8	6	8	7	52

Table 11. Processing time (ms) for each algorithm using 1000 sequences with a minimum support of 75% over synthetic data.

Algorithm		APR	M-SP	MSA	PPA	FP-G	F1/FP	F1
Period	4	17	2.9×10^3	5	4	9	4	6
	8	36	2.7×10^3	6	5	8	4	16
	12	51	2.8×10^3	7	5	6	5	27
	16	71	2.7×10^3	9	6	7	6	44
	20	97	2.8×10^3	10	8	10	8	71

5.2. Results Derived from the Geolife Dataset

We chose data samples from three Geolife users—Users 0, 1, and 2—and we reviewed these samples manually to confirm the presence of periodic patterns. Different size sequences—500, 750, and 1000—and periods—10, 15, 25, 50, and 100—were considered. The 500 dataset was built from User 0's records, the 750 dataset was built from User 1's records, and the 1000 dataset was built from User 2's records. Supports for 25% (Tables 12–14), 50% (Tables 15–17), and 75% (Tables 18–20) were considered.

Table 12. Processing time (ms) for each algorithm using 500 sequences with a minimum support of 25% over real data.

Algorithm		APR	M-SP	MSA	PPA	FP-G	F1/FP	F1
Period	10	55	433	7	8	11	7	47
	15	93	465	8	9	11	8	59
	25	1.3×10^3	620	2.3×10^3	9.4×10^5	16	21	153
	50	-	4.6×10^3	-	-	1305	27	614
	100	-	2.7×10^4	-	-	-	35	1.4×10^3

Table 13. Processing time (ms) for each algorithm using 750 sequences with a minimum support of 25% over real data.

Algorithm		APR	M-SP	MSA	PPA	FP-G	F1/FP	F1
Period	10	91	1.2×10^3	14	14	17	12	47
	15	146	1.2×10^3	16	18	21	15	93
	25	342	1.3×10^3	66	22	23	35	207
	50	3.2×10^4	1.8×10^3	1.1×10^5	8.6×10^5	1.6×10^3	56	670
	100	-	2.9×10^4	-	-	6.9×10^5	80	1.7×10^3

Table 14. Processing time (ms) for each algorithm using 1000 sequences with a minimum support of 25% over real data.

Algorithm		APR	M-SP	MSA	PPA	FP-G	F1/FP	F1
Period	10	93	2.8×10^3	17	17	21	15	57
	15	148	2.9×10^3	18	23	27	20	120
	25	346	3.0×10^3	25	28	34	33	304
	50	-	4.1×10^3	-	-	48	75	1.0×10^3
	100	-	2.0×10^4	-	-	-	94	2.4×10^3

Table 15. Processing time (ms) for each algorithm using 500 sequences with a minimum support of 50% over synthetic data.

Algorithm		APR	M-SP	MSA	PPA	FP-G	F1/FP	F1
Period	10	54	450	7	9	10	7	31
	15	92	467	8	10	11	8	57
	25	189	496	10	12	13	12	122
	50	956	624	29	15	14	27	243
	100	-	1.9×10^3	-	-	-	35	940

Table 16. Processing time (ms) for each algorithm using 750 sequences with a minimum support of 50% over synthetic data.

Algorithm		APR	M-SP	MSA	PPA	FP-G	F1/FP	F1
Period	10	90	1.1×10^3	15	14	17	11	47
	15	143	1.2×10^3	17	17	20	15	93
	25	339	1.3×10^3	21	21	23	19	196
	50	1.7×10^3	1.8×10^3	36	27	31	31	613
	100	1.4×10^4	2.6×10^3	60	31	40	52	1.6×10^3

Table 17. Processing time (ms) for each algorithm using 1000 sequences with a minimum support of 50% over synthetic data.

Algorithm		APR	M-SP	MSA	PPA	FP-G	F1/FP	F1
Period	10	93	2.7×10^3	16	17	22	14	56
	15	147	2.8×10^3	20	23	28	20	120
	25	344	2.9×10^3	25	28	34	32	303
	50	1.5×10^3	3.2×10^3	40	31	38	46	640
	100	-	4.5×10^3	-	-	51	94	1.5×10^3

Table 18. Processing time (ms) for each algorithm using 500 sequences with a minimum support of 75% over synthetic data.

Algorithm		APR	M-SP	MSA	PPA	FP-G	F1/FP	F1
Period	10	55	430	7	8	11	7	31
	15	93	471	8	9	11	8	54
	25	182	494	10	13	13	12	118
	50	766	593	15	15	15	15	187
	100	-	1.1×10^3	-	-	7.0×10^3	35	573

Table 19. Processing time (ms) for each algorithm using 750 sequences with a minimum support of 75% over synthetic data.

Algorithm		APR	M-SP	MSA	PPA	FP-G	F1/FP	F1
Period	10	88	1.1×10^3	14	14	17	12	47
	15	142	1.2×10^3	17	17	20	15	93
	25	337	1.3×10^3	21	21	23	20	196
	50	1.6×10^3	1.6×10^3	35	27	31	31	301
	100	1.2×10^4	2.5×10^3	60	30	40	51	1.4×10^3

Table 20. Processing time (ms) for each algorithm using 1000 sequences with a minimum support of 75% over synthetic data.

Algorithm		APR	M-SP	MSA	PPA	FP-G	F1/FP	F1
Period	10	92	2.7×10^3	16	17	22	15	56
	15	142	2.8×10^3	20	24	27	20	122
	25	343	2.9×10^3	25	28	35	32	302
	50	1.5×10^3	3.1×10^3	40	32	37	45	551
	100	1.3×10^4	4.1×10^3	70	34	39	56	1.3×10^3

6. Discussion

As mentioned previously, timestamped data on the grid are mapped to a character string representing a sequence. All the algorithms which we have included in our research operate on such sequences, and both their performance and scalability depend solely on the sequence's length and the minimum support. Our results are independent of the size of the grid and the configuration of its cells. Thus, the impact of the mapping is not considered here. There is, on the other hand, a separate body of research that studies indexing and searching methods in spatio-temporal databases. These works are based on indexing structures such as the *r-tree* and its variants [42], namely, HR-tree and MVR-tree to mention a couple. Given that such works depend on these structures, it is not possible to compare them directly with the association rule algorithms that we have described here.

6.1. Synthetic Data Results

Tables 3–5 show the experimental results over the synthetic STDB. In these tables, the minimum support was set to 25%. The sequence length in Table 3 is 500, in Table 4 is 750, and in Table 5 is 1000.

Even though our experiments were limited to a maximum of three hours, they shed light on the algorithms' performance. In Table 3, the best results for average processing time are provided by F1/FP (6.4 ms), followed by F1 (26.6 ms), M-SP, FP-G, and PPA. The first results are in line with the standard deviation presented by the first two algorithms—1.85 for F1/FP and 17.02 for F1. The worst results were by APR and MSA. These two algorithms also had the worst standard deviations—2.32×10^7 for APR with an average time of 1.74×10^7 ms, and 9.20×10^6 for MSA with an average time of 9.20×10^6 ms.

Table 4 reflects the same behavior as Table 3, maintaining the same order of less and more efficient algorithms in terms of processing time. According to Table 5, it is possible to see the same performance trend for both the least and most efficient algorithms. Whenever the sequence length was increased in Tables 3 and 4, the processing times also increased for all the algorithms.

Tables 6–8 present the results considering a minimum support of 50%. In Table 6 the sequence length is 500, whereas in Table 7 is 750, and in Table 8 is 1000. In Table 6, the worst performance is by MSA, with an average time of 1.0×10^4 ms and a standard deviation of 3.99×10^3, followed by PPA, whose average time was 1.77×10^3 ms with a standard deviation of 3.52×10^3. Conversely, the best times are provided by F1/FP and F1. F1/FP has an average of 4.8 ms and a standard deviation of 1.16, while F1 has an average of 20.6 ms with a standard deviation of 13.23.

Table 7 exhibits the same behavior as Table 6—that is, the same order of performance for the two most efficient and the two least efficient algorithms. The worst average time in Table 8 was recorded by PPA (2.82×10^3 ms), while its standard deviation was 6.79×10^3. The second worst average—2.82×10^3—was recorded by M-SP, while the second worst standard deviation—4.79×10^3—was presented by MSA.

It is worth noting that the PPA is particularly affected when the period is 12 in Tables 6–8, as both its average time and standard deviations increase. However, the PPA is not the only one affected. All algorithms are impacted negatively by the same period, except for F1/FP and F1. This peculiarity with period 12 could be attributed to how the pattern is formed, as both AP and MSA are not affected as much as the PPA. Following the same trend observed in Tables 3–5, the best average time along with the best standard deviation is yielded by F1/FP and F1.

Tables 9–11 display the results considering a minimum support of 75%, with sequences of lengths 500 (Table 9), 750 (Table 10), and 1000 (Table 11). In Table 9, the worst average time was yielded by M-SP with 442.2 ms, while the second worst was from APR—41.6 ms with a standard deviation of 24.62. Remarkably, the standard deviation for F1 (13.141) was the second worst. The most efficient algorithm was the PPA, with an average of 3.8 ms and a standard deviation of 1.30. The second most efficient one was MSA, whose average time was 4 ms. Note that FP-G registered the lowest standard deviation, with a value of 0.707.

As in the case of Table 9, the less efficient algorithms in Table 10 are M-SP, with an average of 1.22×10^3 ms, and a standard deviation of 44.7. APR presents an average time of 46.8 ms with a standard deviation of 27.36. FP-G exhibits the lowest standard deviation, and the PPA proves to be the most efficient with an average of 4.4 ms and the second-lowest standard deviation of 1.14. F1/FP offers the second-best average—that is, 4.8 ms—and the third-best standard deviation of 1.30. Notably, F1 continues to have better time average and standard deviation than APR and M-SP.

Finally, Table 11 shows the same trend as Tables 9 and 10. The highest standard deviation was displayed by M-SP at 2.78×10^3, and the lowest time average was exhibited by F1/FP at 5.4 ms, followed by the PPA at 5.6 ms. Note that F1/FP presents a high standard deviation, though it is negligible in comparison with the PPA, and F1 has better averages than M-SP and APR.

To summarise, from this set of experiments, we can appreciate that every time the sequence length is increased, the processing time also increases. In addition, whenever the support is raised, all algorithms tend to reduce their average time and their standard deviation, which implies lower processing times for each of them. PPA, MS-P, and FP-G greatly benefit from the minimum support being increased. On the other hand, F1/FP and F1 exhibit a scalable performance that is independent of both increments, the minimum support and the sequence length. This is particularly notable in comparison with the performance of the other algorithms, especially when the support is low.

6.2. Real Data Results

Tables 12–14 display experimental results on the real dataset. In these three tables, the minimum support was set to 25%. Specifically, the sequence lengths are 500, 750, and 100, respectively, for each table. In Table 12, three algorithms exceed the maximum of three hours, particularly when the periods are 50 and 100. These algorithms correspond to APR, MSA, and PPA, which present the highest averages for time along with their corresponding standard deviations—that is, replacing "-" with three hours in milliseconds. According to the results of this table, the most efficient algorithms are F1/FP with an average of 19.6 ms and F1 with 454 ms. Their standard deviations were 12.11 and 577.23, respectively. Table 13 exhibits the same behavior as Table 12, maintaining the same positions for the least efficient algorithms in terms of running time, particularly when the period is 100. Following the same pattern as Table 12, F1/FP and F1 had the lowest averages and standard deviations. Continuing this trend, Table 14 provides the same rankings for the best and worst averages along with their standard deviations.

Four algorithms—APR, MSA, PPA, and FP-G—exceeded the time limit in Table 15. These four algorithms yielded the highest standard deviations. Conversely, the lowest averages and standard deviations corresponded to F1/FP and F1. No algorithm exceeded the time limit in Table 16. However, the highest averages were provided by APR (3.25×10^3 ms with a standard deviation of 6.04×10^3), followed by M-SP with an average of 1.60×10^3 and a standard deviation of 6.20×10^2.

Two algorithms obtained the lowest averages: PPA with 22 ms and a standard deviation of 7, followed by F1/FP with 25.6 ms. As for Table 17, three algorithms exceeded the time limit—APR, MSA, and PPA—when the period was 100. Also, note that these algorithms presented the highest standard deviations. The algorithm with the lowest average was FP-G—34 ms with a standard deviation of 10.99—followed by F1/FP—41.2 ms and a standard deviation of 31.956.

Table 18 is no exception to the fact that some algorithms exceeded the time limit, such as APR, MSA, and PPA, specifically when the period was 100. The lowest averages were given by F1/FP—15.4 ms with a standard deviation of 11.41—and F1—192.6 ms with a standard deviation of 221.14. As for Table 19, no algorithm exceeded 3 h of processing. The highest average times were given by APR with 2.83×10^3 ms and a standard deviation of 5.16×10^3. The second-highest times corresponded to M-SP with an average of 1.54×10^3 ms and a standard deviation of 568.33.

Finally, Table 20 shows that no algorithms exceeded the maximum time limit. The lowest average time was provided by PPA—27 ms with a standard deviation of 6.782 . The second best average time was for FP-G—32 ms with a standard deviation of 7.211.

Just as it happened with the synthetic dataset, every time the support was increased in the real dataset, the running times decreased, except for F1/FP and F1. Similarly, when the sequence length was increased, the running times also increased.

At first glance, the running times are higher on the real dataset than on the synthetic one. However, for both datasets, the running times decreased for the algorithms using minimum support every time the minimum support increased. F1/FP always showed remarkable running times. Indeed, this algorithm was always among the best ones. Incidentally, when the minimum support was increased, PPA and MS-P also achieved good results on the real dataset.

7. Conclusions

Mining periodic patterns became a topic of relevance in the 1990s, mostly after the development of the Apriori algorithm. Since then, the discovery of patterns has turned out to be one of the main techniques for characterizing data. Over the years, several improvements to the basic Apriori idea have been considered, focusing on larger and larger datasets as time has progressed, increasingly stressing the storage and processing capabilities of modern computers.

In this paper, we have presented F1/FP, a new probabilistic algorithm, which is guaranteed to find all the periodic patterns—it always returns the correct answer, as any Las Vegas algorithm. F1/FP does not require minimum support and is scalable.

Given that no previous work has compared the performance of the most well-regarded algorithms in this field, we endeavored to compare them through extensive experimentation, involving sequences of different lengths and various support thresholds. This has enabled us to gain a broader understanding of the performance of the most relevant algorithms, and we have confirmed that our proposal performs better than the existing alternatives. Our experiments allow us to derive the following observations :

F1/FP Performance: F1/FP has a robust performance not only on synthetic data but also on real data. F1/FP provides the best average results compared to all the other algorithms included in our study.

Apriori: The Apriori algorithm achieves the worst results for synthetic and real datasets.

PPA: When support is increased for real data, the PPA has a reasonably good performance. When support is low and data are synthetic, the PPA is not ideal.

MS-Apriori: The performance of MS-Apriori is remarkably good on synthetic data.

FP-Growth The FP-Growth algorithm achieves a better performance than the PPA when support is increased.

Minimum support: Every time the minimum support is increased, all algorithms accomplish better processing times.

Performance on real data: Broadly speaking, all algorithms increase their average times when using real data.

Future Work

Although F1/FP always returns the correct answer and is scalable, we must continue to work on its runtime execution to ensure that it is as fast as possible. To improve the runtime in future research, we plan to exploit parallelism, including what we can obtain through a *MapReduce* formulation [43].

We also want to consider an extension of our work to manage continuous online streams. While our algorithm is guaranteed to handle such a challenge, it would be an interesting case study to utilize it to detect and classify events in real time as they are retrieved from social media and astronomical data streams.

Author Contributions: Conceptualization, C.G.-S.; methodology, C.G.-S.; software, C.G.-S.; validation, C.G.-S., P.G. and M.A.P.; formal analysis, C.G.-S.; investigation, C.G.-S., P.G. and M.A.P.; data curation, C.G.-S.; writing—original draft preparation, C.G.-S., P.G. and M.A.P.; writing—review and editing, M.A.P.; funding acquisition, C.G.-S. and M.A.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research was partially funded by Universidad del Bío-Bío, Chile, under grant INN I+D 23-53. Marco Palomino acknowledges the support provided by the University of Aberdeen to support his work on this publication.

Data Availability Statement: Restrictions apply to the availability of the data employed as part of our investigation. Data were obtained from the Geolife project (*Microsoft Research Asia*) and are available at https://www.microsoft.com/en-us/download/details.aspx?id=52367 (accessed on 28 May 2024) with the permission of Microsoft Research Asia.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Frederick, D.E. Librarians in the Era of Artificial Intelligence and the Data Deluge. *Libr. Tech News* **2020**, *37*, 1–7. [CrossRef]
2. Li, X.; Chen, W.; Chan, C.; Li, B.; Song, X. Multi-Sensor Fusion Methodology for Enhanced Land Vehicle Positioning. *Inf. Fusion* **2019**, *46*, 51–62. [CrossRef]

3. Lü, G.; Batty, M.; Strobl, J.; Lin, H.; Zhu, A.X.; Chen, M. Reflections and Speculations on the Progress in Geographic Information Systems (GIS): A Geographic Perspective. *Int. J. Geogr. Inf. Sci.* **2019**, *2*, 346–367. [CrossRef]
4. Nandal, R. Spatio-Temporal Database and its Models: A Review. *IOSR J. Comput. Eng.* **2013**, *11*, 91–100. [CrossRef]
5. Alhaek, F.; Liang, W.; Rajeh, T.M.; Javed, M.H.; Li, T. Learning Spatial Patterns and Temporal Dependencies for Traffic Accident Severity Prediction: A Deep Learning Approach. *Knowl. Based Syst.* **2024**, *286*, 111406. [CrossRef]
6. Ireland, L.G.; Robbins, J.; Neal, R.; Barciela, R.; Gilbert, R. Generating Weather Pattern Definitions over South Africa Suitable for Future Use in Impact-Orientated Medium-Range Forecasting. *Int. J. Climatol.* **2024**, *44* , 1513–1529. [CrossRef]
7. Nezhadettehad, A.; Zaslavsky, A.; Abdur, R.; Shaikh, S.A.; Loke, S.W.; Huang, G.L.; Hassani, A. Predicting Next Useful Location with Context-Awareness: The State-of-the-Art. *arXiv* **2024**. [CrossRef]
8. Bechini, A.; Bondielli, A.; Dell'Oglio, P.; Marcelloni, F. From Basic Approaches to Novel Challenges and Applications in Sequential Pattern Mining. *Appl. Comput. Intell.* **2023**, *3*, 44–78. [CrossRef]
9. Agrawal, R.; Srikant, R. Fast Algorithms for Mining Association Rules. In Proceedings of the 20th International Conference on Very Large Data Bases (VLDB), Santiago, Chile, 12–15 September 1994; Volume 1215, pp. 487–499.
10. Liu, B.; Hsu, W.; Ma, Y. Mining Association Rules with Multiple Minimum Supports. In Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, 15–18 August 1999; pp. 337–341.
11. Han, J.; Dong, G.; Yin, Y. Efficient Mining of Partial Periodic Patterns in Time Series Database. In Proceedings of the 15th International Conference on Data Engineering (Cat. No. 99CB36337), Sydney, NSW, Australia, 23–26 March 1999; IEEE: Piscataway, NJ, USA , 1999; pp. 106–115.
12. Yang, K.J.; Hong, T.P.; Chen, Y.M.; Lan, G.C. Projection-based Partial Periodic Pattern Mining for Event Sequences. *Expert Syst. Appl.* **2013**, *40*, 4232–4240. [CrossRef]
13. Gutiérrez-Soto, C.; Gutiérrez-Bunster, T.; Fuentes, G. A New and Efficient Algorithm to Look for Periodic Patterns on Spatio-Temporal Databases. *J. Intell. Fuzzy Syst.* **2022**, *42*, 4563–4572. [CrossRef]
14. Mochizuki, Y. apyori 1.1.2. 2019. Available online: https://pypi.org/project/apyori/ (accessed on 18 May 2024).
15. Clarkson, K.L. Las Vegas Algorithms for Linear and Integer Programming when the Dimension is Small. *J. ACM* **1995**, *42*, 488–499. [CrossRef]
16. Jamshed, A.; Mallick, B.; Kumar, P. Deep Learning-Based Sequential Pattern Mining for Progressive Database. *Soft Comput.* **2020**, *24*, 17233–17246. [CrossRef]
17. Bunker, R.; Fujii, K.; Hanada, H.; Takeuchi, I. Supervised Sequential Pattern Mining of Event Sequences in Sport to Identify Important Patterns of Play: An Application to Rugby Union. *PLoS ONE* **2021**, *16*, e0256329. [CrossRef]
18. Parthasarathy, S.; Mehta, S.; Srinivasan, S. Robust Periodicity Detection Algorithms. In Proceedings of the 15th ACM International Conference on Information and Knowledge Management, Arlington, VA, USA, 6–11 November 2006; pp. 874–875.
19. Khanna, S.; Kasurkar, S. Design & Implementation of Efficient Periodicity Mining Technique for Time Series Data. *Int. J. Adv. Res. Comput. Commun. Eng.* **2015**, *4*, 439–444. Available online: https://www.ijarcce.com/upload/2015/may-15/IJARCCE%2095.pdf (accessed on 18 May 2024).
20. Tirumalasetty, S.; Jadda, A.; Edara, S.R. An Enhanced Apriori Algorithm for Discovering Frequent Patterns with Optimal Number of Scans. *arXiv* **2015**. [CrossRef]
21. Ozden, B.; Ramaswamy, S.; Silberschatz, A. Cyclic Association Rules. In Proceedings of the 14th International Conference on Data Engineering, Orlando, FL, USA, 23–27 February 1998; IEEE: Piscataway, NJ, USA, 1998; pp. 412–421.
22. Samoliya, M.; Tiwari, A. On the Use of Rough Set Theory for Mining Periodic Frequent Patterns. *Int. J. Inf. Technol. Comput. Sci.* **2016**, *8*, 53–60. [CrossRef]
23. Kiran, R.U.; Venkatesh, J.; Toyoda, M.; Kitsuregawa, M.; Reddy, P.K. Discovering Partial Periodic-Frequent Patterns in a Transactional Database. *J. Syst. Softw.* **2017**, *125*, 170–182. [CrossRef]
24. Huang, K.Y.; Chang, C.H. SMCA: A General Model for Mining Asynchronous Periodic Patterns in Temporal Databases. *IEEE Trans. Knowl. Data Eng.* **2005**, *17*, 774–785. [CrossRef]
25. Hatkar, S.P.; Kadam, S.; Syed, A. Analysis of Various Periodicity Detection Algorithms in Time Series Data with Design of New Algorithm. *Int. J. Comput. Appl. Technol. Res.* **2014**, *3*, 229–239. [CrossRef]
26. Gao, Y.; Wang, S.; Padmanabhan, A.; Yin, J.; Cao, G. Mapping spatiotemporal patterns of events using social media: A case study of influenza trends. *Int. J. Geogr. Inf. Sci.* **2018**, *32*, 425–449. [CrossRef]
27. Lv, M.; Chen, L.; Chen, G. Mining User Similarity Based on Routine Activities. *Inf. Sci.* **2013**, *236*, 17–32. [CrossRef]
28. Ishida, K. Periodic Topic Mining from Massive Amounts of Data. In Proceedings of the 2010 International Conference on Technologies and Applications of Artificial Intelligence, Hsinchu, Taiwan, 18–20 November 2010; IEEE: Piscataway, NJ, USA, 2010; pp. 379–386.
29. Cheng, T.; Wicks, T. Event Detection Using Twitter: A Spatio-Temporal Approach. *PLoS ONE* **2014**, *9*, e97807. [CrossRef] [PubMed]
30. Atluri, G.; Karpatne, A.; Kumar, V. Spatio-Temporal Data Mining: A Survey of Problems and Methods. *ACM Comput. Surv.* **2018**, *51*, 1–41. [CrossRef]
31. Pillai, K.G.; Angryk, R.A.; Banda, J.M.; Schuh, M.A.; Wylie, T. Spatio-Temporal Co-Occurrence Pattern Mining in Data Sets with Evolving Regions. In Proceedings of the IEEE 12th International Conference on Data Mining Workshops, Brussels, Belgium, 10 December 2012; IEEE: Piscataway, NJ, USA, 2012; pp. 805–812.

32. Pillai, K.G.; Angryk, R.A.; Aydin, B. 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, Orlando, FL, USA, 5–8 November 2013; pp. 104–113.
33. Lahiri, M.; Berger-Wolf, T.Y. Periodic Subgraph Mining in Dynamic Networks. *Knowl. Inf. Syst.* **2010**, *24*, 467–497. [CrossRef]
34. Han, J.; Pei, J.; Yin, Y. Mining Frequent Patterns without Candidate Generation. *ACM Sigmod Rec.* **2000**, *29*, 1–12. [CrossRef]
35. Gutiérrez-Soto, C.; Galdames, P.; Navea, D. Efficiently Finding Cyclical Patterns on Twitter Considering the Inherent Spatio-temporal Attributes of Data. *J. Univers. Comput. Sci.* **2023**, *29*, 4563– 4572. [CrossRef]
36. Xun, Y.; Wang, L.; Yang, H.; Cai, J. Mining Relevant Partial Periodic Pattern of Multi-Source Time Series Data. *Inf. Sci.* **2022**, *615*, 638–656. [CrossRef]
37. Bahmani, B.; Goel, A.; Shinde, R. Efficient Distributed Locality Sensitive Hashing. In Proceedings of the 21st ACM International Conference on Information and Knowledge Management, Maui, HI, USA, 29 October–2 November 2012; pp. 2174–2178.
38. Jiang, Z.; Tang, Y.; Gu, J.; Zhang, Z.; Liu, W. Discovering Periodic Frequent Travel Patterns of Individual Metro Passengers Considering Different Time Granularities and Station Attributes. *Int. J. Transp. Sci. Technol.* 2023, *in press*.
39. Savasere, A.; Omiecinski, E.; Navathe, S. An Effcient Algorithm for Mining Association Rules in Large Databases. In Proceedings of the 21st International Conference on Very Large Databases (VLDB), Zurich, Switzerland, 11–15 September 1995; pp. 432–444.
40. Berberidis, C.; Aref, W.G.; Atallah, M.; Vlahavas, I.; Elmagarmid, A.K. Multiple and Partial Periodicity Mining in Time Series Databases. In Proceedings of the 15th European Conference on Artificial Intelligence (ECAI), Lyon, France, 21–26 July 2002; Volume 2, pp. 370–374.
41. Zheng, Y.; Fu, H.; Xie, X.; Ma, W.Y.; Li, Q. Geolife GPS Trajectory Dataset—User Guide. 2011. Available online: https://www.microsoft.com/en-us/research/publication/geolife-gps-trajectory-dataset-user-guide/ (accessed on 1 June 2024).
42. Arge, L.; Berg, M.d.; Haverkort, H.; Yi, K. The Priority R-tree: A Practically Efficient and Worst-Case Optimal R-tree. *ACM Trans. Algorithms* **2008**, *4*, 1–30. [CrossRef]
43. Hashem, I.A.T.; Anuar, N.B.; Gani, A.; Yaqoob, I.; Xia, F.; Khan, S.U. MapReduce: Review and Open Challenges. *Scientometrics* **2016**, *109*, 389–422. [CrossRef]