CCPM: A Scalable and Noise-Resistant Closed Contiguous Sequential Patterns Mining Algorithm

Yacine Abboud, Anne Boyer, Armelle Brun

CCPM: A Scalable and Noise-Resistant Closed Contiguous Sequential Patterns Mining algorithm

Yacine Abboud[1], Anne Boyer[1], and Armelle Brun[1]

Université de Lorraine, LORIA UMR 7503, France

Abstract. Mining closed contiguous sequential patterns has been addressed in the literature only recently, through the CCSpan algorithm. CCSpan mines a set of patterns that contains the same information than traditional sets of closed sequential patterns, while being more compact due to the contiguity. Although CCSpan outperforms closed sequential pattern mining algorithms in the general case, it does not scale well on large datasets with long sequences. Moreover, in the context of noisy datasets, the contiguity constraint prevents from mining a relevant result set. Inspired by BIDE, that has proven to be one of the most efficient closed sequential pattern mining algorithm, we propose CCPM that mines closed contiguous sequential patterns, while being scalable. Furthermore, CCPM introduces usable wildcards that address the problem of mining noisy data. Experiments show that CCPM greatly outperforms CCSpan, especially on large datasets with long sequences. In addition, they show that the wildcards allows to efficiently tackle the problem of noisy data.

Keywords: Data mining, sequential pattern mining, closed contiguous sequential pattern, scalable, noise resistant

1 Introduction

Pattern mining [2] is one of the most studied topic in the data mining literature. An ordered pattern is usually called a sequential pattern. Many applications rely on sequential patterns: pattern discovery in protein sequences [15], [23], analysis of customer behavior in web logs [6], sequence-based classification [4], etc. Consequently, sequential pattern mining [3] has become a significant part of pattern mining studies. Many sequential pattern mining algorithms have been proposed to solve various issues, such as general sequential pattern mining [12], [18], string mining [9], [16], closed sequential pattern mining [20, 21], constraint-based sequential pattern mining [11], [19], etc.

A pattern is said closed if there is no super-pattern with the same support. Closed sequential mining has become popular as the set of closed patterns extracted contains the same information than the set of general sequential patterns, but with less redundancy [24].

Many real-life tasks greatly benefit from patterns with contiguous items. For example in text mining [10], contiguous patterns are used for statistical natural

language processing or document classification. In Web log mining [6,7], they are used to predict navigation paths and thus to improve the design of web pages. The search of frequent contiguous sequential patterns in DNA and amino acid sequences is unavoidable to reveal common shared functions [13, 14]. The contiguity constraint leads to the extraction of much fewer patterns with a shorter average length, while carrying the same information [24]. That is why closed contiguous sequential patterns (CCSPs) are highly relevant to mine data. However, the added value provided by the contiguity constraint has its cost. In case of noisy data, the support of the mined contiguous patterns is underestimated, making some of these patterns not frequent.

CCSpan is the only algorithm designed to mine CCSPs, its design stands on a snippet-growth scheme to generate candidate patterns. A snippet is a contiguous sub-pattern. CCSpan outperforms, in runtime and memory usage, all the closed sequential pattern mining algorithms in the literature [24]. Nevertheless, it suffers from the same issue than other contiguous pattern mining algorithms: the lack of adaptability to noisy data. In addition, despite being the most efficient algorithm to extract sequential patterns, CCSpan does not scale well on large databases having long sequences.

Two scientific questions thus arise: 1) How to improve contiguous sequential pattern mining algorithms to more accurately extract information from noisy data? 2) How to increase the efficiency of CCSP mining algorithms to scale with large databases and long sequences?

In this paper, we introduce a new algorithm CCPM that overcomes those two issues. CCPM is based on Prefixspan [18] and on BIDE [20], two of the most efficient sequential mining and closed sequential mining algorithms [8]. CCPM mines closed contiguous patterns with usable wildcards. A wildcard [8] is a joker of one item regarding the contiguity constraint. Let $\langle a, b, d, c\rangle$ be a pattern, it can be considered as the pattern $\langle a, b, c\rangle$ with one wildcard. Unlike other algorithms with wildcards of the literature, CCPM introduces a maximal number of wildcards that can be used per pattern. To the best of our knowledge, CCPM is the first noise-resistant CCSP mining algorithm due to the use of wildcards. CCPM also offers the possibility to mine patterns that start with a constrained item and thus gets a new set of patterns not included in the closed contiguous sequential patterns. Experiments show that CCPM greatly outperforms CCSpan, especially on large datasets with long sequences. Hence the main contributions of this paper are:

- we introduce the second closed contiguous sequential pattern mining algorithm that significantly outperforms the first one, CCSpan;
- we use expendabled wildcards to make CCPM noise-resistant;
- we present experiments on standard datasets and on jobs offers datasets to show how CCPM performs on large and noisy data;

This paper is organized as follows. Section 2 introduces concepts and notations required by the formulation of CCSP problem. In Section 3 we explore properties of CCSP and present CCPM. Section 4 is dedicated to the experiments conducted, while Section 5 concludes.

2 Preliminaries and Related Work

2.1 Preliminaries.

In this section we introduce some definitions and notations used for CCSP mining.
Let $I = \{i_1, i_2, ..., i_n\}$ be a set of distinct items. A sequential pattern P is an ordered list denoted as $P = \langle e_1, e_2, ..., e_m \rangle$ where $e_k \in I$ is an item for $1 \leq k \leq m$. For brevity, a sequential pattern is also written $e_1 e_2 ... e_m$. A sequential pattern $P_1 = a_1 a_2 ... a_n$ is contiguously contained in another sequential pattern $P_2 = b_1 b_2 ... b_m$, if $n < m$ and there exist consecutive integers $1 \leq j_1 < j_2 < ... < j_n < m$ such that $a_1 = b_{j_1}, a_2 = b_{j_2}, ..., a_n = b_{j_n}$. P_1 is called a contiguous sub-pattern or a snippet of P_2 and P_2 a contiguous super-pattern of P_1, denoted by $P_1 \sqsubset_c P_2$. The concatenation of P_1 and P_2 is a new contiguous sequential pattern, denoted as $\langle P_1, P_2 \rangle$.
An input sequence database SDB is a set of tuples (sid, S), where sid is a sequence id, and S an input sequence. Table 1 contains an example of a sequence database, made up of 4 tuples and the set of items $I = \{A, B, C\}$. The number of tuples in SDB is called the size of SDB and is denoted by $|SDB|$. The absolute support of a contiguous sequential pattern P_α in a sequence database SDB is the number of tuples in SDB that contiguously contain P_α, denoted by $sup_A^{SDB}(S_\alpha)$. Similarly, the relative support of P_α in SDB is the proportion of sequences in SDB that contain P_α, denoted by $sup_R^{SDB}(S_\alpha)$. The universal contiguous support is the number of occurrences of P_α in SDB, denoted by $uni\_sup^{SDB}(P_\alpha)$. Given a threshold min_sup, a contiguous sequential pattern P_α is frequent in SDB if $sup_A^{SDB}(P_\alpha) \geq min\_sup$ or $sup_R^{SDB}(P_\alpha) \geq min\_sup$. P_α is a closed contiguous sequential pattern if there exists no contiguous sequential pattern P_β such that: $P_\alpha \sqsubset_c P_\beta$, and $sup_A^{SDB}(P_\alpha) = sup_A^{SDB}(P_\beta)$. An item e is a *starting item*, denoted $\_e$, if it satisfies a chosen constraint. A pattern is a *starting item* pattern if its first item is a *starting item*. Aside from the above notations, we further present some definitions.

Table 1: An example of sequence database SDB

Sequence id	Sequence
1	CAABC
2	ABCB
3	CABC
4	ABBCA

Example 1. We present different sets of patterns from the sequence database SDB (shown in Table 1) when $min\_sup = 2$ with one wildcard. The complete set of frequent contiguous sequential patterns consists of 7 sequential patterns $S_{fc} = \{A : 4,\ AB : 4,\ ABC : 3,\ B : 4,\ BC : 4,\ C : 4,\ CA : 3\}$. The set of frequent closed contiguous sequential patterns consists of only 4 sequential patterns $S_{fcc} = \{AB : 4,\ ABC : 3,\ BC : 4,\ CA : 3\}$. The complete set

of frequent contiguous sequential patterns with one wildcard is $S_{fcw} = \{A : 4,\ AB : 4,\ ABB : 2,\ ABC : 4,\ B : 4,\ BB : 2,\ BC : 4,\ C : 4,\ CA : 3,\ CB : 2,\ CAB : 2,\ CABC : 2\}$. The use of one wildcard allows to discover the sequential patterns ABB, BB, CB, $CABC$ and to increase the support of ABC by one, making it a frequent CCSP. The set of frequent CCSPs with one wildcard is thus $S_{fccw} = \{ABB : 2,\ ABC : 4,\ CA : 3,\ CB : 2,\ CABC : 2\}$. The pattern ABC has absorbed AB and BC.

2.2 Related Work.

The sequential pattern mining problem was first introduced by Agrawal and Srikant in [3] with the Apriori algorithm. Apriori is based on the monotonicity property "all nonempty subsets of a frequent itemset must also be frequent" [2]. Since then, many sequential pattern mining algorithms have been proposed for performance improvements: Prefixspan [18], SPAM [5], etc. Prefixspan algorithm uses projected databases to mine frequent sequential patterns. Prefixspan recursively extends a prefix sequential pattern by adding a frequent item from the projected database of this prefix. This new representation of data, based on projected databases, allows a more efficient mining [5] and thus Prefixspan performs really well regarding execution time. As closed sequential patterns lead to a more compact set of patterns but also a better efficiency, several algorithms focus their mining on such patterns [17], [22]. Clospan [21] is one of them, it mines a candidate set of closed patterns based on Prefixspan algorithm, and keeps the candidate patterns set for post-pruning. The candidate set is expensive in term of memory usage. To cope with this issue, the BIDE algorithm [20] does not memorize any candidate set of closed patterns. BIDE generates new patterns with the framework of Prefixspan and uses a BI-Directional closure checking scheme that does not use any set of candidates, hence improves greatly both runtime and memory usage. The philosophy of this scheme is to check if a frequent pattern can be extended to the left (backward extension) or to the right (forward extension) with at least one item. The pattern is closed if it can not be extended in each sequence. BIDE is known to be one of the most efficient closed sequential pattern mining algorithm [8]. One major issue in closed sequential pattern mining comes from large databases: the set of frequent closed sequential patterns becomes unmanageable. To solve this problem, CCSpan [24] algorithm introduced closed contiguous sequential pattern (CCSP) mining. Frequent CCSPs are far fewer and shorter than closed sequential patterns. CCSpan uses a snippet-growth scheme to generate candidate patterns. The original sequence is split into a set of snippets of the same length and their support is calculated to keep the frequent ones as candidate patterns. Then for each candidate pattern, two sub-patterns are considered, the one without the first item of the pattern and the one without the last item. The closure checking is performed recursively by comparing the support of the pattern and the support of both sub-patterns. Despite being more efficient than closed sequential pattern mining algorithms due to the contiguity constraint, CCSpan scales rather poorly on datasets with long sequences. Indeed, the framework that builds snippets becomes highly time consuming as sequences become longer. Furthermore, CCSpan suffers from a lack

of adaptability of CCSPs to noisy data. In fact, the support of CCSPs on noisy data may not reflect the real support of those patterns in the data. Therefore, the CCSPs can not be used to accurately extract information from noisy data. The issues raised by the state-of-the art make us consider the following questions: (1) Can the highly efficient framework from Prefixspan that mines frequent sequential patterns, used as well in Clospan and BIDE, be used to get frequent CCSPs? (2) How to take inspiration from BIDE, one of the most efficient algorithm for closed sequential patterns mining, to mine CCSPs? (3) How to bypass CCSpan algorithm issues with noisy data and keep the added value of contiguity?

3 CCPM: Mining Closed Contiguous Sequential Patterns

In this section, we introduce the CCPM algorithm (Closed Contiguous sequential Patterns Mining), which shares the philosophy of both Prefixspan [18] and BIDE algorithms [20]. The use of contiguous sequential patterns mining algorithm is only relevant in case of clean data but most of the time data still contains noise. In order to overcome this issue of noisy data, we introduce a maximal number of wildcards per pattern. As the mined data is usually just a bit noisy, the number of wildcards allowed is often rather small. In addition, CCPM allows to mine patterns starting with an item that satisfies a given constraint: *starting item* patterns. Such patterns are interesting in many domains. In text mining, patterns starting with an action verb in order to identify competencies [1] or in marketing, purchase patterns starting by a given category of items to establish more targeted selling strategies, etc. As a contiguous sequential *starting item* pattern (CSSP) can only start with a *starting item*, a closed contiguous sequential *starting item* pattern (CCSSP) is necessarily equal or included in a closed contiguous sequential super-pattern. If it is included, the CCSSP will not appear in the set of mined CCSPs. Therefore, if one CCSSP is not equal to the closed contiguous sequential super-pattern then the set of CCSSPs is not included in the set of CCSPs.
Therefore, CCPM uses a framework that enumerates frequent contiguous sequential *starting item* patterns with wildcards (CSSPWs), inspired by the framework of Prefixspan to enumerate frequent sequential patterns. CCPM uses a closure scheme and a pruning technique inspired by the BI-Directional closure scheme and the Backscan method of BIDE to mine closed patterns. CCPM relies on three steps listed here: First, a framework that mines frequent CSSPWs. Second, a BI-Directional checking scheme, adapted to CSSPWs, that forms closed contiguous sequential *starting item* patterns with wildcards (CCSSPWs). Third, a contiguous pruning techniques used to improve the efficiency of CCPM.

3.1 Framework to enumerate frequent CSSPWs.

Here we introduce some definitions about projected contiguous items and database in order to explain the framework.

Definition 1. ***Projected contiguous items of a prefix sequential pattern with wildcards:*** *Given an input sequence S and a sequential pattern P with*

*$P \sqsubset_c S$, the projected contiguous items of P are all the adjacent items after each occurrence of P in S and P is called a prefix sequential pattern. For example, in the sequence ABBCA of Table 1, the projected contiguous items of the prefix sequential pattern B are B and C. If a wildcard can be used, all the second adjacent items after P are projected items too. A wildcard is denoted by * on the item it corresponds to. On the previous example, the item A is a projected contiguous item of B with a wildcard, noted A*. A wildcard is not used if the first and the second adjacent items after P are the same. For example, with the prefix sequential pattern A, the projected contiguous items are B and B*, so we only keep B. To summarize, the projected contiguous items with a wildcard of the prefix sequential pattern B in ABBCA are $\{B, C, A^*\}$.*

Definition 2. ***Projected contiguous database of a prefix sequential pattern with wildcards:*** *The complete set of projected contiguous items in SDB of a prefix sequential pattern P with wildcards is called the projected contiguous database of P with wildcards.*

Theorem 1 (Projected database pruning). *Given two items e_1 and e_2, if e_2^* is always a projected contiguous item of P together with e_1, and if e_2 is never a projected contiguous item, e_2^* can be safely removed from the projected contiguous database.*

Proof. Given two items e_1 *and* e_2, a prefix sequential pattern P and a sequence database SDB. If e_2^* is always a projected contiguous item of P together with e_1 and if e_2 is never a projected contiguous item of P in SDB, it means that there exists a contiguous sequential pattern $Q = \langle P, e_1, e_2 \rangle$ with $uni\_sup^{SDB}(Q) = uni\_sup^{SDB}(\langle P, e_2 \rangle)$. Thus, the sequential pattern $\langle P, e_2 \rangle$ is always included in Q and so is its expansions. Therefore, it is useless to expand it.

The complete search space of contiguous sequential patterns forms a sequential pattern tree [5]. The root node of the tree is at the top level and is labeled $\emptyset$. The framework of CCPM recursively extends a node (referred to as a prefix sequential pattern) at a certain level in the tree by adding a contiguous item, resulting in a child node at the next level. As CCPM mines *starting item* patterns, the framework only extends frequent items satisfying the constraint. Applying this constraint greatly reduces the search space, hence greatly improves its efficiency. It also improves the results by removing irrelevant patterns. When all frequent *starting items* are found, each one becomes a prefix sequential pattern to expand, they are at the first level of the tree. The framework then builds the projected contiguous database of each prefix. To know if a wildcard is avalaible, the count of wildcards used is memorized for each occurrence of the prefix sequential pattern. Then, only the frequent items in the projected contiguous database grow the prefix sequential pattern.
Now we describe how to enumerate frequent closed contiguous sequential patterns with one wildcard on this example. The prefix sequential pattern A is the first studied, its projected contiguous database is: $\{A, B, C^*;\ B, C^*;\ B, C^*;\ B\}$. In this projected database, B has a support of 4 and C a support of 3. As their

support exceed min_sup, both of them can be extended. However, B always occurs together with C^* in the projected contiguous database and C is never a projected contiguous item. According to Theorem 1, A can be extended with B, C can be removed. Then, given the prefix sequential pattern AB, the projected database is: $\{C;\ C, B^*;\ C;\ B, C^*\}$. C has a support of 4 and B a support of 2, both of them can be extended. The projected database of ABC is: $\{\emptyset;\ B;\ \emptyset;\ A\}$ and the one of ABB is: $\{\emptyset;\ C\}$. As no item have a support greater or equal to $min\_sup = 2$, the prefix sequential pattern A is now completely extended. The same procedure can be used with B and C.

3.2 BI-Directional contiguous closure checking with wildcards.

The previous framework mines frequent CSSPWs. To get the set of frequent CCSSPWs, a closure checking has to be used. CCPM closure checking scheme is based on the same BI-Directional design than BIDE. The definitions of backward and forward extensions of BIDE are adapted to CSSPWs with the same number of wilcards for all CSSPWs.

Given two CSSPWs $P_1 = \_b_1 b_2 ... b_n$ and $P_2 = \_a_1 ... a_m \_b_1 b_2 ... b_n$ in a sequence database SDB. If $sup_A^{SDB}(P_1) = sup_A^{SDB}(P_2)$, $\langle \_a_1 ... a_m \rangle$ is called a backward extension of P_1. The specificity of CSSPW forces the backward extension to be a *starting item* pattern. Given a sequence S of SDB and a CSSPW P with $P \sqsubset_c S$. For each occurrence of P in S, the backward space of P is the CSSPW Q such that $Q = \_c_1 ... c_j$ and $\langle Q, P \rangle \sqsubset_c S$ where $\_c_1$ is the first *starting item* on the left of this occurrence of P in S. If there is no *starting item* on the left of P, P can not have a backward extension. Let $P_3 = \_b_1 b_2 ... b_n d_1 ... d_k$ be a CSPW, if $sup_A^{SDB}(P_1) = sup_A^{SDB}(P_3)$, we say $d_1 ... d_k$ is a forward extension item of P_1.

With those definitions in mind, the following theorem and lemma are straightforward.

Theorem 2. *A CSSPW P is closed, if and only if P has no backward extension and no forward extension items.*

Lemma 1. *Given a CSSPW P in a sequence database SDB, if there is a backward space Q of P in SDB, with* $sup_A^{SDB}(\langle Q, P \rangle) = sup_A^{SDB}(P)$*, then Q is a backward extension of P.*

Lemma 2. *For a given CSSPW P, its complete set of forward extensions items is the set of items in its projected contiguous database that have a support equal to* $sup_A^{SDB}(P)$.

Proof. For each item e_k in the projected contiguous database of P, a new CSSPW $P_k = \langle P, e_k \rangle$ can be created, if $sup_A^{P\_SDB}(e_k) = sup_A^{SDB}(P)$ thus $sup_A^{SDB}(P_k) = sup_A^{SDB}(P)$ then $\langle e_k \rangle$ is a forward extension of P.

We introduce two stop conditions during the backward spaces scanning to improve the efficiency of the backward extension checking. The backward checking extension of a CSSPW is stopped as soon as: 1) a *starting item* cannot be found in the backward spaces of a sequence; 2) one of the two backward contiguous items (if a wildcard is available) or the backward contiguous item (no wildcard) of the prefix sequential pattern cannot be found in the backward spaces of a sequence.

3.3 The contiguous BackScan pruning.

To prune the useless parts of the search space, we propose to use the Backscan technique from BIDE, adapted to CCSSPW mining. The idea behind Backscan is to detect if the current prefix sequential pattern can be absorbed by a prefix sequential pattern that will be mined later. If it is the case, it is not expanded.

Theorem 3 (The Backscan pruning). *Given a CSSPW P in SDB, if there exists a backward space Q with $uni\_sup(\langle Q, P\rangle) = uni\_sup^{SDB}(P)$, P can be safely pruned.*

Proof. Given a CSSPW P and a backward space Q with $uni\_sup(\langle Q, P\rangle) = uni\_sup^{SDB}(P)$. Assume we extend P with a contiguous sequential pattern K to get a longer CSSPW R ($R = \langle P, K\rangle$), we can also construct another CSSPW $R' = \langle Q, P, K\rangle$. As $uni\_sup^{SDB}(\langle Q, P\rangle) = uni\_sup^{SDB}(P)$, we have $uni\_sup^{SDB}(R) = uni\_sup^{SDB}(R')$. Hence, we can not use P as a prefix sequential pattern to generate any CCSSPW.

3.4 The CCPM algorithm.

CCPM (Algorithm 1), integrates the three previous steps: the framework that searches frequent CSSPWs, the BI-Directional closure checking and the BackScan pruning.

Algorithm 1 CCPM(SDB, min_sup, W)

Input:a sequence database SDB, a minimum support threshold min_sup, a maximal number of wildcards W
Output:F, the complete set of frequent CCSSPWs

1: $F = \{\emptyset\}$
2: $F1 = frequent\_starting\_items(SDB, min\_sup)$;
3: **foreach** $f1$ in $F1$ do
4: $\quad f1\_SDB = projected\_database(SDB, f1, W)$;
5: $\quad$ **if** (!**BackScan**$(f1)$)
6: $\quad\quad$ call **pGrowth**$(f1\_SDB, f1, min\_sup, F)$;
7: **return** F;

Algorithm 2 pGrowth(P_SDB, P, min_sup, F)

Input: projected sequence database with the used wildcards P_SDB, a prefix sequential pattern P, a minimum support threshold min_sup
Output:F, the current set of frequent CCSSPWs

1: $FI = frequent\_items(P\_SDB, min\_sup)$;
2: $BEI =$ **Backward_Check**(P, P_SDB, W)
3: **if** (!**BackScan**(P)
4: $\quad FEI = \left|\{z \text{ in } FI \mid z.sup = sup^{SDB}(P)\}\right|$;
5: $\quad$ **if** $(\{BEI \cap FEI\} == \{\emptyset\})$
6: $\quad\quad F = F \bigcup \{P\}$;
7: $\quad$ **foreach** i *in* FI do
8: $\quad\quad P^i = \langle P, i\rangle$;
9: $\quad\quad P^i\_SDB = projected\_database(P\_SDB, P^i, W)$;
10: $\quad\quad$ call **pGrowth**$(P^i\_SDB, P^i, min\_sup, F)$;

The input parameters of Algorithm 1 are a sequence database, a minimum support and a maximal number of wildcards per pattern. CCPM starts by scanning the database to find the set of frequent *starting items*: $F1$ (line 2). The *starting items* constraint can be removed without any impact on CCPM. The *projected_database* function (line 4) creates the projected database of each item in $F1$ and specifies if a wildcard has been used or not for each occurrence of each item. *BackScan* (line 5) is applied to check if a frequent item can immediately be pruned. The subroutine pGrowth (Algorithm 2) is called for each frequent *starting item* with its projected database. pGrowth checks if the input pattern P is closed and recursively extends P with each frequent items in its projected database. pGrowth continuously updates the set of CCSSPWs. The function *Backward_Check* (line 2) looks for backward extensions, with 2 stop conditions designed to improve the efficiency. *Backscan* (line 3) is used again to check if the prefix can be pruned, then if there are no forward and no backward extensions (line 4 and 5), the input pattern P is added to the set of frequent CCSSPW: F (line 6). Then the projected database of P is scanned to find the set of frequent contiguous items and specifies if a wildcard has been used or not for each item (line 9). Each frequent item extends P (line 8) and becomes the prefix sequential pattern to recall pGrowth until CCPM returns F, the complete set of frequent CCSSPW.

4 Experiments

We conduct a comprehensive performance study to evaluate CCPM. [24] has shown how CCSpan outscales and outperforms closed sequential pattern mining algorithms such as Clospan and BIDE. Therefore, we only compare CCPM to our implemantation of CCSpan to evaluate its efficiency. First, CCPM is compared to CCSpan [24] in terms of running time and memory usage. The parameters of CCPM are set so that CCSpan and CCPM correspond to the same configuration: CCSP mining. Second, CCPM with no constraints and CCPM with *starting item* patterns and wildcards are compared to evaluate the impact of these constraints.

4.1 Datasets and environnement.

The experiments are performed on a i7-4750HQ 2GHz, 8GB memory on Windows 10. In order to evaluate the performance of CCPM, we use five real datasets that cover a wide range of distribution characteristics. Two of them are reference datasets used to evaluate sequential pattern mining algorithms. $BMS1$ *Gazelle* [21], [20], [24] used in the KDD-CUP 2000 competition and $Mushroom$, available online on the SPMF website[1]. The three other datasets are datasets we formed from online job offers, so made up of noisy textual data. The $BMS1$ *Gazelle* dataset contains clickstream and purchase data from Gazelle.com, a legwear and legcare web retailer. $BMS1$ *Gazelle* is sparse. The second dataset, $Mushroom$, is composed of characteristics of various species

[1] http://www.philippe-fournier-viger.com/spmf/index.php

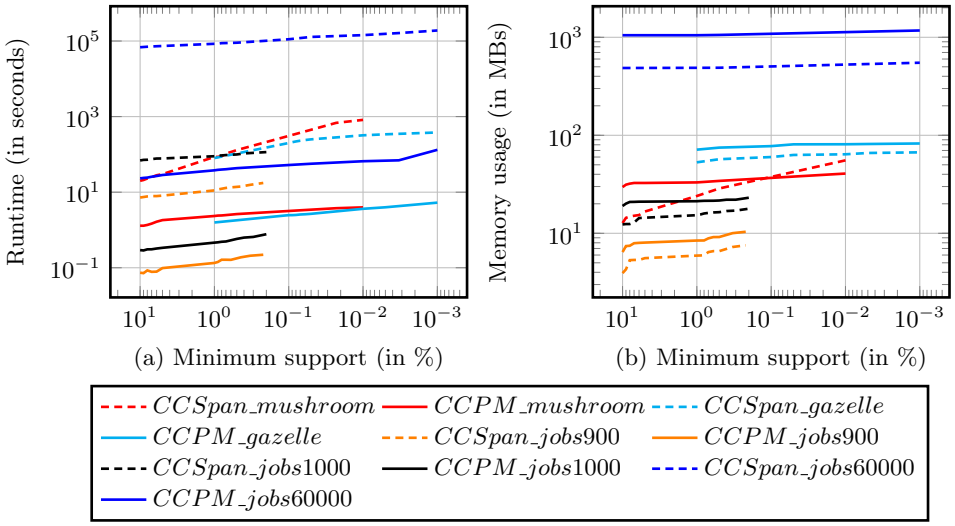

Fig. 1: Runtime(a) and Memory(b) on `Mushroom, Gazelle, Jobs900, Jobs1000 and Jobs60000` datasets

of mushrooms and was originally obtained from the UCI repository of machine learning databases. *Mushroom* is dense. The three job datasets (*jobs*900, *jobs*1000 and *jobs*60000) are composed of job offers collected on websites between January 2016 and June 2016. They reflect the kind of textual data that can be found on the web, they are noisy and very sparse. The characteristics of the five datasets are shown in Table 2. Some of these datasets share similarities on some attributes, so we can easily compare the influence of each attribute on CCPM. The datasets *BMS*1 *Gazelle* and *jobs*60000 as well as *jobs*900 and *jobs*1000 have approximately the same number of sequences with a very different average length, they will be compared to evaluate the impact of average length. The datasets *jobs*900 and *Mushroom* as well as *jobs*1000 and *jobs*60000 have the same average length, they will be compared to evaluate the impact of the number of sequences. The experiment will highlight the impact of both average length and number of sequences on the efficiency of CCSpan and CCPM.

Table 2: Datasets characteristics

Dataset	#seq.	#items	avg.len.
*BMS*1 *Gazelle*	59,601	497	2.4
Mushroom	8,416	119	23.0
*jobs*900	900	4,460	26.6
*jobs*1000	1,000	8,813	76.0
*jobs*60000	60,000	92,394	77.0

4.2 CCSP mining: comparison of CCSpan and CCPM.

As CCPM can be set to mine CCSPs like CCSpan, we can conduct a series of experiments on the five datasets in terms of execution time and memory usage, with various relative support thresholds (Figure 2). Each line stops when the absolute support reaches the value of 1 for the corresponding dataset. First of all, Figure 2 shows that neither CCSpan nor CCPM are significantly impacted by the variation of the support. Indeed, both runtime and memory usage are never increased by more than 10 whereas the number of patterns is increased from 110 times on $jobs1000$ to 6,200 on $jobs60000$. The only exception is the runtime of CCSpan on the $Mushroom$ dataset, which is increased by almost 100. The contiguity constraint seems to make CCSpan and CCPM almost independent on the support and on the number of patterns mined. As the variation of the support has little impact on runtime and memory usage, we choose to set the support value for the rest of the experiments, this value is set to $min\_sup = 1\%$.

Average length We first evaluate the impact of the average length of sequences on the execution time of CCSpan and CCPM (Figure 2a). $jobs900$ and $jobs1000$ share a similar number of sequences (900 vs 1,000) with a very different average length (23 vs 76). Same with $BMS1$ $Gazelle$ and $jobs60000$ (59,601 vs 60,000) that also differ in the average sequence length (2.4 vs 77). CCSpan runs 8 times faster on $jobs900$ than on $jobs1000$ whereas the average length of $jobs900$ is 3 times smaller (ratio of 2.7). Same reasoning between $BMS1$ $Gazelle$ and $jobs60000$, CCSpan runs 6,800 faster on $BMS1$ $Gazelle$ than on $jobs60000$ whereas the average length is 32 times smaller (ratio of 212). We can conclude that the average length of the sequences exponentially impacts the runtime of CCSpan.
CCPM runs 3.5 times faster on $jobs900$ than on $jobs1000$ (ratio of 1.2) and 25 times faster on $Gazelle$ than on $jobs60000$ (ratio of 0.8). We can conclude that, contrary to CCSpan, the runtime of CCPM is linearly impacted by the length of the sequences.

Number of sequences Now we focus on the impact of the number of sequences on the execution time of CCSpan and CCPM. $jobs900$ and $Mushroom$ share a similar average length of sequences (26.6 vs 23), while having a different number of sequences (900 vs 8,416). The two datasets $jobs1000$ and $jobs60000$ also have a similar average length of sequences (76 vs 77), while having a very different number of sequences (1,000 vs 60,000). CCSpan runs 8 times faster on $jobs900$ than on $Mushroom$ whereas the number of sequences of $jobs900$ is 9 times smaller (ratio of 0.9). CCSpan runs 915 times faster on $jobs1000$ than on $jobs60000$ whereas the number of sequences of $jobs1000$ is 60 times smaller (ratio 15). We can conclude that the number of sequences exponentially impact the runtime of CCSpan. This result has to be tempered due to specificity of the $Mushroom$ dataset, indeed it is the only dataset that impacts the runtime of CCSpan, while having a support that varies.
CCPM runs 18 times faster on $jobs900$ than on $Mushroom$ (ratio of 2) and

83 times faster on $jobs1000$ than on $jobs60000$ (ratio of 1.4). We observe that, contrary to CCSpan, the number of sequences linearly impacts the runtime of CCPM.
Finally, we compare the runtime between CCSpan and CCPM on all datasets. CCPM always greatly outperforms CCSpan in execution time on all the datasets and no matter the support. The ratio is going from 15 on $BMS1$ $Gazelle$ to 2,175 on $jobs60000$.
We can conlude that CCPM is only linearly impacted by the number of sequences and their average length and is always significantly faster than CCSPan.

Memory usage We now focus on the evaluation of the memory usage of CCSpan and CCPM (Figure 2b). CCSpan is always more efficient than CCPM in terms of memory usage, except on the $Mushroom$ dataset. We observe that the ratio between CCSpan and CCPM, ranges from 1.25 on $BMS1$ $Gazelle$ to 2.1 on $jobs60000$. The difference both algorithms remains rather small, CCSpan and CBIDE are in the same order of magnitude, regarding the memory usage. CCSpan has an average increase of memory usage of 1.8 with the support variation through all datasets. CCPM has an average increase of memory usage of 1.2 with support variation on all datasets. We observe that both algorithms are really stable; we suppose that it is another added value of the contiguity constraint.
This experiment has shown that when mining CCSPs the support variation has almost no impact on both runtime and memory usage, probably due to the contiguity constraint. In addition, the memory usage is extremely stable. CCPM is always significantly faster than CCSpan, regardless the dataset and the support, while having memory usage in the same order of magnitude. Moreover, the runtime of CCPM is linearly impacted by the number of sequences and their average length, contrary to CCSpan. We have thus answered the second scientific question: CCPM scales well on large databases having long sequences.

Table 3: runtime, patterns of CCPM with constraints

conf.	pattern	runtime	len>=3	activities
no con.	14,574	49	2,947	882
1WC	26,655	142	7,603	2,019
2WC	29,508	164	10,473	2,489
3WC	31,817	176	12,782	2,711
SI	3,828	27	1,365	1,365
SI&1WC	7,938	42	3,597	3,597
SI&2WC	9,757	49	5,417	5,417
SI&3WC	11,649	55	7,308	7,308

4.3 CCSSPW evaluation.

In the previous experiment, CCPM was implemented with no constraint for the sake of comparison with CCSpan. In this section, we show how the wildcards

and the *starting item* patterns impact CCPM.
This experiment is conducted on the *jobs*60000 dataset with a relative support of 0.1% (in order to mine a significant amount of patterns and perform a relevant analysis). In the employment sector, activities are at the core of job offers. An activity is a coherent set of completed tasks organized toward a predefined objective with a result that can be measured. An activity is formalized by action verbs [1]. In this context, activities are patterns starting with a verb with a length greater or equal to 3. The *starting item* constraint is thus defined as the *verb* category. "Inform clients by explaining procedures", "Manage business by performing related duties" or "Start operations by entering commands" are examples of activities.

We study runtime and activities of CCPM with *starting item* (SI) and/or wildcards (WC) and compare them to those of CCPM with no constraint. Table 3 shows the results of this experiment. The memory usage (not shown in Table 3) remains stable (about 1000 MBs), whatever is the configuration, the memory used is thus independent of the configuration.
CCPM with no constraint mines 14,574 patterns in 49 seconds. 2,947 of them have a length greater or equal to 3 and 882 of them start with a verb, thus CCPM with no constraint mines 882 activities. We observe that CCPM with the *starting item* (SI) constraint mines 3,828 patterns in 27 seconds and 1,365 of them have a length greater or equal to 3. Therefore, CCPM with *starting item* mines 3.8 less patterns but the the execution time is only divided by 2. In addition, the SI constraint allows to find 65% more activities. Those two results are the consequence of the backward checking with *starting item* patterns. Indeed, instead of looking for any item, the function is looking for the first *starting item* on the left of the pattern, thus taking more time to execute the backward extension checking. In addition, a *starting item* pattern can have a non *starting item* closed super-pattern. In this case, the list of patterns mined by a closed contiguous sequential mining algorithm does not contain the *starting item* pattern. This is the reason why the set of activities mined with the *starting item* constraint is larger than the set of activities mined without any constraint.
Now we evaluate the impact of wildcards on CCPM with and without the *starting item* constraint. We observe that the use of one, two and three wildcards increases the number of activities mined by respectively 230%, 280% and 307% for CCPM with no *starting item* and of 260%, 400% and 535% with *starting item*. Those numbers show that the use of wildcards greatly impacts the number of activities found and confirm that *jobs*60000 is actually a noisy dataset. Logically, introducing the wildcard constraint also increases the runtime. Recall that the wildcard constraint does not impact the memory usage. Going from no wildcard to one, two and three wildcards, the median and the mode of the length of the activities mined are respectively: 4 & 3; 6 & 3; 11 & 3; 16 & 3. The median length increases with the number of wildcards, but the mode remains stable. An expert of the domain stated that an activity is averagely made up of 3 to 6 words [1]. With 2 wildcards, the median reaches 11, patterns are becoming too long. Therefore, the optimal number of wildcards to

mine activities in this dataset seems to be 1.
To conclude, this experiment shows that the *starting item* constraint enables to mine a new set of CCSSPs not included in the set of CCSPs. Additionally, this constraint greatly decreases the execution time of CCPM. This experiment also shows that *jobs*60000 is a noisy dataset and that the optimal configuration of CCPM exploits the *starting items* constraint and 1 wildcard. So, we answered the first scientific question raised in the introduction: CCPM is a way to more accurately mine patterns in noisy data.

5 Conclusion

In this paper, we proposed a novel algorithm for mining frequent closed contiguous sequential patterns: CCPM. CCPM can rely on usable wildcards to fit noisy data while preserving the added value of contiguity. The experiments showed that CCPM greatly outperforms CCSpan in execution time, while being competitive in memory usage. In addition, to the best of our knowledge, CCPM is the first algorithm able to efficiently mine closed contiguous sequential patterns in large datasets with long sequences. Hence, CCPM is a solution to the two scientific questions raised in the introduction: 1) How to improve the adaptability of contiguous sequential pattern mining algorithms to more accurately extract information from noisy data? 2) How to increase the efficiency of CCSP mining algorithms to scale with large databases and long sequences? The *starting items* constraint allows to mine a new set of patterns not included in the set of closed contiguous sequential patterns.
In a future work, CCPM will be evaluated on several others datasets with various characteristics. We will also focus on the improvement of the memory usage of CCPM. We will also work on enriching the types of constraint (on multiple items or on a item not necessarily at the start of the pattern) without increasing the execution time.

References

1. Abboud, Y., Boyer, A., Brun, A.: Predict the emergence - Application to competencies in job offers, ICTAI (2015)
2. Agrawal, R., Imieliskiand, T., Swami, A.: Mining Association Rules Between Sets of Items in Large Databases, ACM SIGMOD, 207–216 (1993)
3. Agrawal, R., Srikant, R.: Mining Sequential Patterns, Proceedings of the Eleventh International Conference on Data Engineering, 3–14 (1995)
4. C. Aggarwal, C., Ta, N., Wang, J., Feng, J., J. Zaki, M.: Xproj: A Framework for Projected Structural Clustering of Xml Documents, Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 46–55 (2007)
5. Ayres, J., Flannick, J., Gehrke, J., Yiu, T.: Sequential Pattern Mining Using a Bitmap Representation, Proceedings of the Eighth ACM SIGKDD International Conference, 429–435 (2002)

6. Chen, J., Cook, T.: Mining Contiguous Sequential Patterns from Web Logs, Proceedings of the 16th International Conference on WWW (2007)
7. Chen, J.: Contiguous item sequential pattern mining using UpDown Tree, Intelligent Data Analysis, 12 (2008)
8. Li, C., Wang, J.: Efficiently Mining Closed Subsequences with Gap Constraints, Proceedings of SIAM International Conference on Data Mining (2008)
9. Fischer, J., Heun, V., Kramer, S.: Optimal string mining under frequency constraints, In European Conference on Principles of Data Mining and Knowledge Discovery, 139–150 (2006)
10. Fürnkranz, J.: A Study Using n-gram Features for Text Categorization, Austrian Research Institute for Artificial Intelligence (1998)
11. Garofalakis, M., Rastogi, R., Shim, K.: MSPIRIT: Sequential Pattern Mining with Regular Expression Constraints, Proceedings of the 25th International Conference on Very Large Data Bases (1999)
12. Han, J., Pei, J., Mortazavi-Asl, B., Chen, Q., Dayal, U., Hsu, M.: FreeSpan: Frequent Pattern-projected Sequential Pattern Mining, Proceedings of the Sixth ACM SIGKDD (2000)
13. Kang, T. H., Yoo, J. S., Kim, H.Y.: Mining Frequent Contiguous Sequence Patterns in Biological Sequences, 2007 IEEE 7th International Symposium on BioInformatics and BioEngineering (2007)
14. Karim, M., Rashid, M., S. Jeong, B., J. Choi, H.: An Efficient Approach to Mining Maximal Contiguous Frequent Patterns from Large DNA Sequence Databases, Genomics Inform, 10 (2012)
15. Liao, V. C. C., Chen, M. S.: DFSP: a Depth-First SPelling algorithm for sequential pattern mining of biological sequences, Knowledge and Information Systems, 38 (2014)
16. Matsui, T., Uno, T., Umemori, J., Koide, T.: A New Approach to String Pattern Mining with Approximate Match, Discovery Science, 110–125 (2013)
17. Pei, J., Han, J., Mao, R., Chen, Q., Dayal, U., Hsu, M.: CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets, In DMKD01 workshop (2001)
18. Pei, J., Han, J., Mortazavi-Asl, B., Chen, Q., Dayal, U., Hsu, M.: PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth, Proceedings of the 17th International Conference on Data Engineering (2001)
19. Pei, J., Han, J., Wang, W.: Constraint-based Sequential Pattern Mining: The Pattern-growth Methods J. Intell. Inf. Syst., 28 (2007)
20. Wang, J., Han, J.: BIDE: Efficient Mining of Frequent Closed Sequences, Proceedings of the 20th International Conference on Data Engineering (2004)
21. Yan, X., Han, J., Afshar, R.: Clospan: mining closed sequential patterns in large datasets, Proceedings of SIAM Conference on Data Mining (2003)
22. Zaki, M., Hsiao, C.: CHARM: An efficient algorithm for closed itemset mining, Proceedings of SIAM Conference on Data Mining, 2 (2002)
23. Zhang, M., Kao, B., Cheung, D., Yip, K.: Mining Periodic Patterns with Gap Requirement from Sequences, ACM Trans. Knowl. Discov. Data, 1 (2007)
24. Zhang, J., Wang, Y., Yang, D.: CCSpan: Mining closed contiguous sequential patterns, Knowledge-Based Systems, 89 , 1–13 (2015)