Knowl Inf Syst (2016) 48:429–463
DOI 10.1007/s10115-015-0884-x

REGULAR PAPER

CloFAST: closed sequential pattern mining using sparse and vertical id-lists

Fabio Fumarola[1] · **Pasqua Fabiana Lanotte**[1] · **Michelangelo Ceci**[1] · **Donato Malerba**[1]

Received: 11 August 2014 / Revised: 20 July 2015 / Accepted: 5 October 2015 /
Published online: 20 October 2015

Abstract Sequential pattern mining is a computationally challenging task since algorithms have to generate and/or test a combinatorially explosive number of intermediate subsequences. In order to reduce complexity, some researchers focus on the task of mining *closed* sequential patterns. This not only results in increased efficiency, but also provides a way to compact results, while preserving the same expressive power of patterns extracted by means of traditional (non-closed) sequential pattern mining algorithms. In this paper, we present CloFAST, a novel algorithm for mining closed frequent sequences of itemsets. It combines a new data representation of the dataset, based on *sparse id-lists* and *vertical id-lists*, whose theoretical properties are studied in order to fast count the support of sequential patterns, with a novel one-step technique both to check *sequence closure* and to *prune* the search space. Contrary to almost all the existing algorithms, which iteratively alternate itemset extension and sequence extension, CloFAST proceeds in two steps. Initially, all closed frequent itemsets are mined in order to obtain an initial set of sequences of size 1. Then, new sequences are generated by directly working on the sequences, without mining additional frequent itemsets. A thorough performance study with both real-world and artificially generated datasets empirically proves that CloFAST outperforms the state-of-the-art algorithms, both in time and memory consumption, especially when mining long closed sequences.

Keywords Sequential pattern mining · Closed sequences · Data mining · Itemset

✉ Michelangelo Ceci
michelangelo.ceci@uniba.it

Fabio Fumarola
fabio.fumarola@uniba.it

Pasqua Fabiana Lanotte
pasquafabiana.lanotte@uniba.it

Donato Malerba
donato.malerba@uniba.it

[1] Department of Computer Science, University of Bari "A. Moro", Via Orabona 4, 70125 Bari, Italy

1 Introduction

Since its introduction [1], sequential pattern mining has become a fundamental data mining task with a large spectrum of applications, including Web mining [15], classification [9], finding copy-paste bugs in large-scale software code [16] and mining motifs from biological sequences [21].

In sequential pattern mining, input data are a set of sequences, called *data sequences*. Each data sequence is an ordered list of *transactions*, where each transaction is a set of literals, called *itemset*. Typically, the order of transactions in the list is based on the time-stamp associated with each transaction, although other non time-related orderings are possible. The output of sequential pattern mining is sequential patterns, each of which consists of a list of items. The problem is to find all sequential patterns with a user-specified minimum *support* (or *frequency*), which is defined as the percentage of data sequences that contain the pattern.

If compared with the more common problem of frequent pattern mining, sequential pattern mining is computationally challenging because, when solving this problem, a combinatorially explosive number of intermediate subsequences have to be generated and/or tested [13]. In fact, although algorithms presented in the literature are relatively efficient [2,18,20,24,25], when they are used to mine long sequences, time and space scalability becomes increasingly critical. This is especially true for low values of the support threshold.

To alleviate this problem, research in sequential pattern mining has made progress in two directions: (i) efficient methods for mining only the set of closed sequential patterns and (ii) efficient methods for pruning the search space and exploiting specifically designed data structures.

As for (i), many studies pinpoint the idea that for mining frequent sequential patterns, one should not mine *all* the frequent sequences [11,17,22,23]. In particular, they propose mining the *closed* sequential patterns, where a sequential pattern α is closed if it has no proper supersequence β with the same support. Intuitively, since all the subsequences of a frequent sequence are also frequent, mining the set of closed sequential patterns may help avoid the generation of unnecessary subsequences, thus leading to more compact results and saving computational time and space costs.

As for (ii), many algorithms avoid maintaining the set of already generated closed sequences during the mining process [23]. Pruning of the search space and closure checking typically exploit multiple pseudo-projected databases [22] (i.e., databases of sequences generated from a single sequence prefix), which are designed to be efficiently queried. However, pseudo-projected databases require significant time and space to be created and queried, thus limiting not only the capability of the algorithms to mine large datasets with long data sequences, but also the capability of the algorithm to process dense data sequences (i.e., data sequences whose itemsets contain many items).

Several approaches (e.g., ClaSP [12] and SPADE [25]) attempt to overcome the limits of pseudo-projected databases by exploiting a vertical representation formalism. However, they all start with 1-itemset sequences and extend them by iteratively alternating *sequence extension*, i.e., appending an itemset to a sequence, and *itemset extension*, i.e., adding an item to an itemset in the sequence. In this way, a frequent itemset mining step is required at each iteration, with a computational cost that does not scale well with the size of frequent sequences.

In this paper, we propose CloFAST (**Clo**sed **FAST** sequence mining algorithm based on sparse id-lists), a novel algorithm to mine closed sequences from large databases of long sequences. It extends and revises the algorithm FAST [19] that extracts only frequent

sequences. In particular, CloFAST, similarly to FAST, combines a new data representation of the dataset (*sparse id-list* and *vertical id-list* [19]) to fast count the support of sequential patterns. However, differently from FAST, it exploits the properties of *sparse id-lists* and of *vertical id-lists*, in order to define a novel one-step technique for *sequence closure checking* and *search space pruning*. Similarly to BIDE [22], CloFAST, during the mining process, does not need to maintain the set of already mined closed sequences [23] to prune the search space and to check whether newly discovered frequent sequential patterns are closed.

CloFAST does not build pseudo-projected databases and does not need to scan them. The initial dataset of sequences of transactions is read once for all to create both *sparse id-lists* and *vertical id-lists*, which are two distinct indexes loaded in the main memory. Sparse id-lists store the position of the transactions which contain a given itemset, while vertical id-lists store the position of a given sequential pattern in the input sequences.

CloFAST uses sparse id-lists to mine closed frequent itemsets and to enumerate the search space, while it uses vertical id-lists to generate the closed sequence patterns. The support of itemsets and sequences is efficiently computed from the sparse id-lists and the vertical id-lists, without requiring additional database scans. Moreover, in order to check the (non-)closure of a considered sequential pattern α and to consequently prune the search space, we propose a novel technique, called *backward closure checking*, which checks whether a new sequence pattern β, obtained by adding a new item/itemset at any position (not necessarily at the end) in α, has the same support as α. In this case, α cannot be considered closed.

Finally, CloFAST mines closed frequent itemsets only at the beginning of the mining process, in order to obtain an initial set of sequences. New sequences are then generated by directly working on the sequences, without generating frequent itemsets.

The contributions of this paper are the following:

1. We propose a two-step process that performs (i) closed itemset mining, and (ii) closed sequential pattern discovery. The two steps only work on sparse id-lists and vertical id-lists, thus gaining efficiency both in time and space.
2. We study formal properties of sparse id-lists and vertical id-lists, which can be used for closed sequential pattern mining.
3. We propose an efficient *backward closure checking* which works on sparse id-lists and vertical id-lists.
4. We present a new *pruning method*, performed during the backward closure checking, which removes non-promising enumerations during the generation of closed sequential patterns.
5. We theoretically prove the correctness and completeness of closed sequential patterns generated by both CloFAST with the backward closure checking technique and CloFAST with pruning.
6. We present empirical evidence that CloFAST outperforms competing algorithms on several real-world and artificially generated sequence datasets.

The rest of the paper is organized as follows. In Sect. 2, the problem of closed frequent sequence mining is defined. Related work is introduced in Sect. 3. Sections 4 and 5 focus on the data structures used to enumerate the search space and for efficient support counting. The CloFAST algorithm and the vertical id-list pruning method are described in Sect. 6. Experimental results and their related discussion are reported in Sect. 7. Finally, conclusions are drawn and future work is outlined.

2 Problem definition and background

Let us consider a sequence database *SDB* of customer transactions. In particular, a sequence represents the (ordered) list of transactions associated with a customer and each transaction consists of a set of items purchased. Each sequence is uniquely identified by a sequence identifier (*sequence-id* or *SID*), while each transaction in the sequence is uniquely identified by a transaction identifier (*transaction-id* or *TID*). The *size* of *SDB* ($|SDB|$) corresponds to the number of sequences (i.e., the number of customers) in the sequence database. In Table 1, we report an example of SDB with three sequences (i.e., $|SDB| = 3$): the first sequence contains five transactions, the second sequence contains two transactions, while the third sequence contains three transactions.

More formally, let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of distinct items, which can be sorted according to some lexicographic ordering $\leq_l$ (e.g., alphabetic ordering). A customer sequence S is a list of transactions, $S = \langle t_1, t_2, \ldots, t_m \rangle$, where each $t_j \subseteq I$ denotes the set of items bought in the jth transaction. The *size* $|\alpha|$ of a sequence α is the number of itemsets (transactions) in the sequence. A sequence $\alpha = \langle a_1, a_2, \ldots, a_m \rangle$ is a **subsequence** of a sequence $\beta = \langle b_1, b_2, \ldots, b_n \rangle$, if and only if integers $i_1, i_2, \ldots, i_m$ exist, such that $1 \leq i_1 < i_2 < \cdots < i_m \leq n$ and $a_1 \subseteq b_{i_1}, a_2 \subseteq b_{i_2}, \ldots, a_m \subseteq b_{i_m}$. We say that β is a **supersequence** of α or that β *contains* α.

Example 1 The sequence $\beta = \langle \{a, b\}, \{c\}, \{d, e\} \rangle$ is a supersequence of $\alpha = \langle \{a\}, \{d\} \rangle$ because $\{a\}$ is a subset of $\{a, b\}$ and $\{d\}$ is a subset of $\{d, e\}$. On the contrary, β is not a supersequence of $\lambda = \langle \{c, d\} \rangle$, since the itemset $\{c, d\}$ is not contained in any itemset of β.

Given a sequence β, its **absolute support** in *SDB* is the number of sequences in *SDB* which contain β, while its **relative support** is the absolute support divided by $|SDB|$. Henceforth, $\beta : s$ will denote the sequence β and its absolute support s, and the term support will refer to the absolute support, unless otherwise specified.

Given two sequences β and α, if β is a supersequence of α and their absolute (or relative) support in SDB is the same, we say that β **absorbs** α. A sequential pattern α is **closed** if no proper sequence β that absorbs α exists.

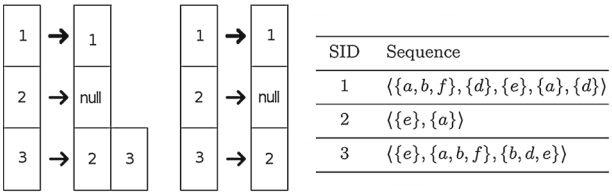
The problem of closed sequence mining is formulated as follows: Given a sequence database *SDB* and a minimum support threshold min_sup, find all the closed sequential patterns in *SDB*, such that their support in *SDB* is at least min_sup. Generated patterns are called **closed frequent** sequential patterns.

Example 2 Table 1 shows an example of a sequence database. If min_sup $= 2$, the complete set of closed frequent sequences consists of only four sequences: $\langle \{a, b, f\}, \{d\} \rangle : 2$, $\langle \{a, b, f\}, \{e\} \rangle : 2$, $\langle \{e\}, \{a\} \rangle : 3$, $\langle \{e\}, \{a\}, \{d\} \rangle : 2$, while the total number of frequent sequences is 26.

The algorithm proposed in this work uses two data structures, called *sparse id-list* (SIL) and *vertical id-list* (VIL), recently introduced in [19] for frequent sequence mining. They

Table 1 Example of a sequence database (SDB)

SID	Sequence
1	$\langle \{a, b, f\}, \{d\}, \{e\}, \{a\}, \{d\} \rangle$
2	$\langle \{e\}, \{a\} \rangle$
3	$\langle \{e\}, \{a, b, f\}, \{b, d, e\} \rangle$



SID	Sequence
1	$\langle\{a,b,f\},\{d\},\{e\},\{a\},\{d\}\rangle$
2	$\langle\{e\},\{a\}\rangle$
3	$\langle\{e\},\{a,b,f\},\{b,d,e\}\rangle$

Fig. 1 From *left to right* **a** the sparse id-lists for itemset $\{b\}$, **b** the sparse id-lists for itemset $\{a, b\}$, **c** the database of sequences

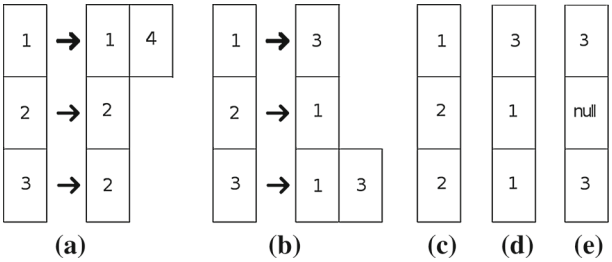


Fig. 2 **a** *sparse* id-list for the itemset $\{a\}$; **b** *sparse* id-list for the itemset $\{e\}$; **c** *vertical* id-list for the sequence $\langle\{a\}\rangle$; **d** *vertical* id-list for the sequence $\langle\{e\}\rangle$; **e** *vertical* id-list for the sequence $\langle\{a\},\{e\}\rangle$

are an optimized representation of the database, since their size is bound by the size of the input dataset. The concept of *id-list* was first introduced by SPADE [2], where an *id-list* of a sequence α was defined as the list of all input customer-id and transaction-id pairs containing α in the database. In the following, we formally introduce them.

Let *SDB* be a sequence database of size n (i.e., $|SDB| = n$) and $S_j \in SDB$ the jth customer sequence ($j \in \{1, 2, \ldots, n\}$).

Definition 1 (*Sparse id-list*) Given an itemset $t \subseteq 2^I$, its *sparse id-list*, denoted as SIL_t, is a vector of size n, such that for each $j = 1, \ldots, n$

$$SIL_t[j] = \begin{cases} \text{the list of the ordered transaction-ids of } t \text{ in } S_j & \text{if } S_j \text{ contains } t \\ null & \text{otherwise} \end{cases}$$

Example 3 Figure 1a shows the SIL_a and $SIL_{a,b}$ of the itemsets $\{a\}$ and $\{a, b\}$, respectively. The values represent the position of the relative itemset in the database in Table 1. Other examples of SILs for the same database are reported in Fig. 2a, b.

Definition 2 (*Vertical id-list*) Given a sequence α, whose last itemset is i, its *vertical id-list*, denoted as VIL_α, is a vector of size n, such that for each $j = 1, \ldots, n$

$$VIL_\alpha[j] = \begin{cases} \text{the transaction-id of } i \text{ in the first occurrence of } \alpha \text{ in } S_j & \text{if } S_j \text{ contains } \alpha \\ null & \text{otherwise} \end{cases}$$

Example 4 Figure 2c–e show some VILs. In particular, Fig. 2e shows the VIL_α of the sequence $\alpha = \langle\{a\},\{e\}\rangle$. Values in VIL_α represent the ending position of the first occurrence of the sequence α in the sequences S_j of Table 1. In particular, the first element (value 3) represents the position of the first occurrence of $\{e\}$, after $\{a\}$ ($\{e\}$ is the last itemset in α), in the first sequence. The second element is null since α is not present in the second sequence.

The third element (value 3) represents the position of the first occurrence of $\{e\}$ (after $\{a\}$) in the third sequence.

3 Related work

To the best of our knowledge, CloSpan [23], BIDE [22], ClaSP [12] and COBRA [14] represent the state of the art in closed sequential pattern mining. CloSpan is based on the *candidate maintenance and test approach*, which generates a candidate set for closed sequential patterns, enumerates the search space and then performs post-pruning. It uses the *equivalence of projected databases* to stop the search and prune the search space. The basic idea is that if a sequence β is a supersequence of a discovered sequence α and the number of items in the corresponding projected databases is the same, then the projected databases are equal and it is possible to stop the search of any descendant of α, since both α and β have the same support.

Wang et al. [22] proposed BIDE as an alternative solution which has the advantage of avoiding candidate maintenance. They presented the *bidirectional extension* schema to generate closed sequences and *BackScan* to prune the search space. The bidirectional extension schema is based on the idea that a sequence $\alpha = \langle a_1, a_2, \ldots, a_m \rangle$ is not closed if an item/itemset a' exists such that it can be used to extend α to a new sequence β, having the same support as α. In particular, β can be obtained from α through either a *forward extension* (adding a new item/itemset after a_m) or a *backward extension* (adding a new item/itemset before a_j, with $1 \leq j \leq m$). If no such item/itemset exists, then α is closed. BIDE does not keep track of any candidate closed sequential patterns for sequence closure checking. This means that it needs multiple scans of the projected databases for both the bidirectional closure checking and the BackScan pruning.

Both CloSpan and BIDE adopt the PrefixSpan [18] approach in the mining phase. PrefixSpan is a pattern-growth divide-and-conquer algorithm that grows sequences by itemset extension and sequence extension. In particular, PrefixSpan grows a prefix pattern to obtain longer sequential patterns by building and scanning its projected database. Although frequent sequences in the projected databases are enumerated to reduce computational complexity, its time complexity is strictly related to the size of the projected databases. For databases with long sequences and large transactions, discovering the local frequent itemsets for each projected database could become an expensive process.

These limitations have been overcome by both SPADE [25] and SPAM [2], which work on more efficient data structures. Improvements are obtained by using a vertical database/bitmap representation (*id-lists*) of the database for both itemsets and sequences. In this way, both itemset extension and sequence extension steps are executed by joining/ANDing operations between vertical/bitmap representation of sequence candidates. Experimental results presented in [2,12] show that both SPAM and SPADE outperform PrefixSpan on *large* datasets, because they avoid the Prefixspan cost for local frequent itemset mining.

The approach used by SPADE has been recently extended in ClaSP [12] for closed sequential pattern mining. In particular, ClaSP exploits the concept of a vertical database format to obtain closed sequences without making several scans of the input database. According to the authors, this significantly improves performances over existing algorithms such as CloSpan. Drawing inspiration from this observation, we decided to exploit both sparse and vertical id-lists (SILs and VILs) to fast count the support of sequential patterns in CloFAST. Contrary to SPADE and ClaSP, where the large size of the id-lists negatively affects the computational

time of the joins, in CloFAST both the itemset extension and the sequence extension are based on SILs and VILs, which can be efficiently used in support counting, sequence closure checking, and search space pruning (see Sect. 6) without performing temporal joins.

Note that all previously referenced algorithms follow the same enumeration strategy: patterns are generated on the basis of the lexicographic ordering and this ordering is then used both in item extension and in sequence extension. However, in general, this pattern-growth strategy may present two drawbacks: redundant itemset extension and expensive "matching cost" in the generation of projected databases.

To explain the first drawback (redundant itemset extension), we report a simple example. Consider a database of two sequences:

$$SDB = [\ \langle\ \{a, b\},\ \{a, b, c\},\ \{a, b\}\ \rangle,\ \langle\ \{a, b, c\}, \{a, b\}, \{a, b\}\rangle].$$

In this case, finding the closed sequence $\langle\ \{a, b\},\ \{a, b\},\ \{a, b\}\rangle$ generally requires three item extensions of $\{a\}$ with $\{b\}$ and three sequence extensions which add $\{a\}$ to the sequence. Graphically, the following steps are typically necessary:

$$\mapsto \langle\ \{a\}\ \rangle \rightarrow \langle\ \{a, b\}\ \rangle \mapsto \langle\ \{a, b\},\ \{a\}\ \rangle \rightarrow \langle\ \{a, b\},\ \{a, b\}\ \rangle \mapsto \langle\ \{a, b\},\ \{a, b\},\ \{a\}\ \rangle$$
$$\rightarrow \langle\ \{a, b\},\ \{a, b\},\ \{a, b\}\ \rangle$$

where $\rightarrow$ indicates the itemset extension and $\mapsto$ indicates the sequence extension. However, if we discover that item $\{a\}$ is not closed (since $\{a, b\}$ absorbs $\{a\}$), then we can directly perform *sequence* extensions of $\{a, b\}$, instead of generating *item* extensions of $\{a\}$. This means that only the following operations are necessary:

$$\mapsto \langle\ \{a, b\}\ \rangle \mapsto \langle\ \{a, b\},\ \{a, b\}\ \rangle \mapsto \langle\ \{a, b\},\ \{a, b\},\ \{a, b\}\ \rangle$$

Obviously, this requires a preliminary closed frequent itemset mining step.

The second drawback (expensive matching cost) is due to queries on (previously generated) projected databases, in order to obtain, after pattern-growth, new projected databases. This process in not trivial since we are working on databases of sequences and a query means a complete scan of the previously generated projected database. Moreover, it is noteworthy that both itemset extension and sequence extension require the generation of a new projected database.

COBRA attempts to overcome these two drawbacks. Instead of extending a pattern by iteratively alternating (i) itemset extension and (ii) sequence extension, it separates the two phases and generates closed frequent itemsets before mining closed sequential patterns. Sequences are extended by only performing sequence extension. Therefore, the closed sequence mining is composed of three consecutive phases: (i) search for all closed frequent itemsets; (ii) transformation of the original dataset into a horizontal format (similar to projected databases); (iii) enumeration of closed sequential patterns.

It is noteworthy that this approach is not equivalent to mining all closed frequent itemsets, then encoding different itemsets as different symbols and finally applying any (non-closed) sequence pattern mining algorithm (*à la* AprioriAll [1], for sequential pattern mining). Indeed, the notions of supersequence/subsequence used to identify closed sequences are based on the notions of superset/subset of itemsets, which cannot be evaluated after encoding. Consequently, the enumeration of closed sequential patterns cannot be based only on input closed itemsets, but it requires additional information extracted during the phase of mining closed itemsets.

CloFAST follows the same approach as COBRA. The difference is that COBRA generates all the sequences of the same length and then performs an expensive post-pruning (called

ExtPruning) to discard non-closed sequences, while CloFAST applies an *online* (i.e., during the sequence generation phase) pruning strategy which operates on vertical id-lists. Moreover, the computation of the pattern support in COBRA requires the identification of the first occurrence of the itemset in each sequence, while in CloFAST it is performed by simply counting the non-null elements in the *vertical id-list* of the pattern. This means that COBRA has to analyze sequences, whereas CloFAST does not.

4 The closed itemset enumeration tree and the closed sequence enumeration tree

In this section, we present the two main data structures used in CloFAST, that is, the closed itemset enumeration tree (CIET) and the closed sequence enumeration tree (CSET). The former is used to store closed frequent itemsets, while the latter is used to store the closed frequent sequential patterns. Similar to the *lexicographic sequence tree* introduced in CloSpan [23], we assume that a lexicographic ordering $\leq_l$ exists in the set of items I. This ordering, as explained in [23], can be extended for sequences composed of itemsets, by exploiting the concepts of sub/superset and sub/supersequence (see Sect. 2). For the sake of simplicity, we will use the same notation $\leq_l$ for this extension of the ordering.

4.1 Closed itemset enumeration tree (CIET)

Similar to a *set enumeration tree* [26], the CIET is an in-memory data structure that allows us to enumerate the complete set of closed frequent itemsets. It is characterized by the following properties: (1) each node in the tree corresponds to an itemset, and the root is the empty itemset ($\emptyset$); (2) if a node corresponds to an itemset i, its children are obtained by itemset extensions from i; and (3) the left sibling of a node precedes the right sibling in the lexicographic order (see Fig. 3 for an example).

Formally, this tree structure is defined as follows:

- the root node of the tree is labeled with $\emptyset$;
- the first level enumerates the frequent 1-item itemsets (i.e., itemsets with a single item in I) according to the ordering $\leq_l$;
- for other levels, nodes represent frequent k-item itemsets, with $k > 1$. Each node is constructed by merging the itemset of its parent node with the itemset of a sibling of its parent node.

Only nodes for (candidate) closed itemsets are added to the CIET. Inspired by the classification of the nodes in Moment [8], we label each node in the CIET as:

- *intermediate*: the node represents a subset of a closed itemset represented in one of its descendant nodes;
- *unpromising*: the node represents a subset of a closed itemset represented in other branches of the tree;
- *closed*: a node is labeled as closed if it represents a closed itemset.

Figure 3 shows an example of a CIET for the database in Table 1, when min_sup $= 2$. Each node contains a frequent itemset and its corresponding support. CloFAST traverses the CIET in a depth-first search order. Only the descendants of the nodes labeled as *closed* or *intermediate* are explored. Indeed, descendants of an unpromising node can be pruned since they cannot represent additional closed itemsets. To check whether or not a certain node

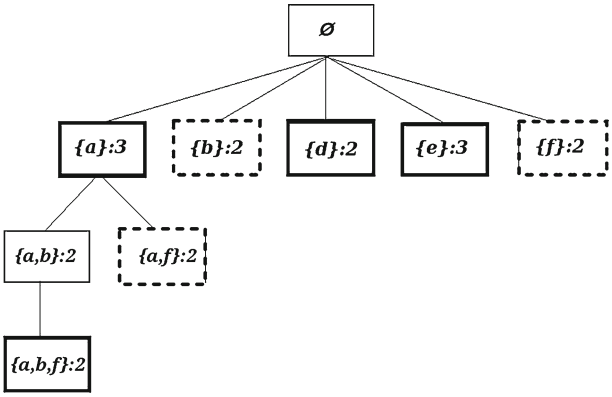


Fig. 3 CIET for our running example. Nodes with *thick borders* represent closed itemsets. Nodes with *dashed borders* represent unpromising nodes. The remaining nodes represent intermediate nodes

corresponding to an itemset i should be labeled as *unpromising*, CloFAST needs to know whether there is a frequent itemset j, such that j absorbs i but does not descend from i. For this purpose, a hashmap (i.e., a structure that maps keys to values) is used to store the set of the closed frequent itemsets associated with a support value, which represents the key of the hashmap. It is noteworthy that nodes labeled as closed can be changed to intermediate during the tree construction.

4.2 Closed sequence enumeration tree (CSET)

The mined set of closed itemsets is used in the construction of the CSET, which enumerates the complete search space of closed sequences, similarly to the sequence tree described in [19]. For the CSET, it is possible to define the following properties: 1) each node in the tree corresponds to a sequence, and the root corresponds to the *null* sequence (Λ) and 2) if a node corresponds to a sequence s, its children are obtained by a sequence extension of s.

This tree has the following structure:

- the root node of the tree is labeled with Λ;
- nodes at the first level represent *candidate* **closed sequences of size 1**, whose unique element is either (i) a closed frequent itemset corresponding to a node labeled as closed in the CIET or (ii) an itemset labeled as intermediate in the CIET for which its SIL is different from the SIL of its closed descendant node;
- nodes at higher first levels represent **sequences of size greater than 1**. Each node can be constructed in two ways: (i) by adding to the sequence of its parent node u the last itemset of the sequence in a sibling of u and (ii) by adding to the sequence of its parent node u the last itemset of the sequence in u itself. The latter guarantees that sequences containing multiple repeated occurrences of the same item/itemset are not discarded (e.g., $\langle\{a, b, f\}, \{a, b, f\}\rangle$ in Example 5). In any case, only nodes for frequent and (candidate) closed sequences are added to the tree.

According to the previous definition, two sibling nodes of a CSET correspond to two distinct sequences of itemsets, $\alpha = \langle a_1, a_2, \ldots, a_m\rangle$ and $\beta = \langle b_1, b_2, \ldots, b_m\rangle$, such that $a_m \neq b_m$ and $\forall i = 1, \ldots, m-1 : \ a_i = b_i$.

Each node in the closed sequence enumeration tree can be labeled as: (i) *closed*, (ii) *non-closed* and (iii) *pruned*.

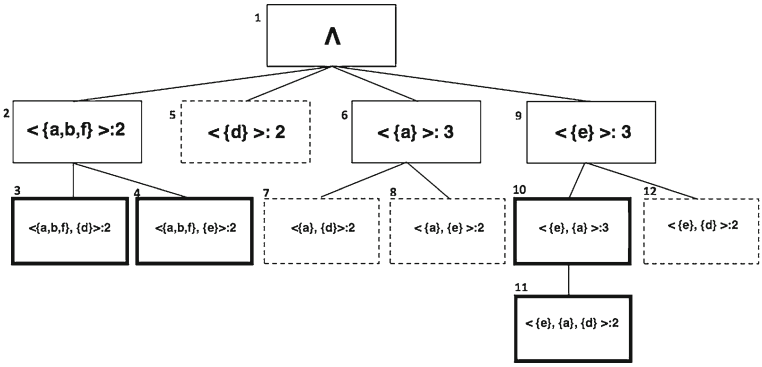


Fig. 4 CSET for our example. Nodes with *thick borders* represent (candidate) closed sequences. Nodes with *dashed borders* represent pruned nodes. Remaining nodes represent non-closed sequences

Figure 4 shows an example of CSET for the database in Table 1 with min_sup = 2. Each node in the figure contains a frequent sequence and its corresponding support. Different borders (thick, dashed or plain) are used for different labeled nodes.

CloFAST builds the CSET in a depth-first search order. Each node in the CSET is considered for sequence extension. In order to exemplify how nodes at the second and at subsequent levels are constructed, we report the following example:

Example 5 Consider the sequence extension of node 2 in Fig. 4. In this case, the candidate sequences are: $\langle \{a, b, f\}, \{d\} \rangle, \langle \{a, b, f\}, \{a\} \rangle, \langle \{a, b, f\}, \{e\} \rangle, \langle \{a, b, f\}, \{a, b, f\} \rangle$. Obviously, not all of them are frequent sequences and are added to the CSET.

5 Properties of SILs and VILs for efficient mining of closed sequential patterns

In this section, we present several properties of VIL and SIL data structures which can be profitably exploited by the sequential pattern mining algorithm.

Proposition 1 *Let* $\alpha = \langle a_1, \ldots, a_m \rangle$*, such that* $VIL_\alpha[j] \neq null$*. Then, for each* i*=1, . . .,* $m - 1$*,* $VIL_{\langle a_1,\ldots,a_i \rangle}[j] < VIL_{\langle a_1,\ldots,a_i,a_{i+1} \rangle}[j]$*.*

Proof It follows from VIL definition. □

Proposition 2 *Let* $\alpha = \langle a_1, \ldots, a_i \rangle$*,* $\epsilon = \langle a_{i+1}, \ldots, a_m \rangle$*,* $VIL_{\alpha\epsilon}[j] \neq null$*. Then,* $VIL_\alpha[j] \neq null$*.*

Proof It follows from the VIL definition. □

These two propositions express two necessary conditions on the VIL structure, when the jth sequence in *SDB* contains α or the composed sequence $\alpha\epsilon$.

Proposition 3 *Let* $\alpha = \langle a_1, \ldots, a_i \rangle, \epsilon = \langle a_{i+1}, \ldots, a_m \rangle, VIL_{\alpha\epsilon}[j] \neq null$*,* γ *any sequence. If* $VIL_\gamma[j] = VIL_\alpha[j]$*, then* $VIL_{\gamma\epsilon}[j] \neq null$*.*

Proof Let $p = VIL_\gamma[j] = VIL_\alpha[j]$, $\langle b_1, \ldots, b_p, b_{p+1}, \ldots, b_r \rangle$, $r \geq m$, be the jth subsequence in *SDB*. From the VIL definition, it follows that the subsequence $\langle b_1, \ldots, b_p \rangle$ contains both α and γ. Moreover, the subsequence $\langle b_{p+1}, \ldots, b_r \rangle$ contains ϵ. Therefore, the jth sequence in *SDB* also contains $\gamma\epsilon$, i.e., $VIL_{\gamma\epsilon}[j] \neq null$. □

This proposition expresses an important property related to the containment of composed sequences. If the jth sequence in *SDB* contains α, $\alpha\epsilon$ and γ, and the position of the last itemset in α coincides with the position of the last itemset in γ, then the jth sequence also contains $\gamma\epsilon$.

Proposition 4 *Let* $\alpha = \langle a_1, \ldots, a_i \rangle$, $VIL_\alpha[j] \neq null$, $\gamma = \langle a_1, \ldots, a_{i-1}, b \rangle$, $VIL_\gamma[j] \neq null$, $\beta = \langle a_1, \ldots, a_{i-1}, b, a_i \rangle$. *If* $VIL_\gamma[j] < VIL_\alpha[j]$, *then* $VIL_\beta[j] = VIL_\alpha[j]$.

Proof It is obvious from the VIL definition and construction of β. □

Proposition 4 states that if the jth sequence of the *SDB* contains two sequences of size i, say α and γ, which differ only in the last itemset, then $VIL_\gamma[j] < VIL_\alpha[j]$ is a sufficient condition to prove that the jth sequence also contains the extended sequence β of size $i + 1$, obtained by juxtaposing a_i to γ.

Proposition 5 *Let* $\alpha = \langle a_1, \ldots, a_i \rangle$, $\epsilon = \langle a_{i+1}, \ldots, a_m \rangle$, $\gamma = \langle a_1, \ldots, a_{i-1}, b \rangle$, $VIL_\gamma[j] \neq null$, $\beta = \langle a_1, \ldots, a_{i-1}, b, a_i, a_{i+1}, \ldots, a_m \rangle$. *If* $VIL_{\alpha\epsilon}[j] \neq null$ *and* $VIL_\gamma[j] < VIL_\alpha[j]$, *then* $VIL_\beta[j] \neq null$.

Proof From proposition 2 and $VIL_{\alpha\epsilon}[j] \neq null$, it follows that $VIL_\alpha[j] \neq null$. Since the conditions for proposition 4 hold, it follows that $VIL_{\langle a_1, \ldots, a_{i-1}, b, a_i \rangle}[j] = VIL_\alpha[j]$. From proposition 3, it follows that $VIL_\beta[j] \neq null$. □

Proposition 6 *Let* $\alpha = \langle a_1, \ldots, a_i \rangle$, $\epsilon = \langle a_{i+1}, \ldots, a_m \rangle$, $\gamma = \langle a_1, \ldots, a_{i-1}, b \rangle$, $a_i \subset b$, $\beta = \langle a_1, \ldots, a_{i-1}, b, a_{i+1}, \ldots, a_m \rangle$. *If* $VIL_{\alpha\epsilon}[j] \neq null$ *and* $VIL_\gamma[j] = VIL_\alpha[j]$, *then* $VIL_\beta[j] \neq null$.

Proof It follows straightforwardly from proposition 3. □

For the sake of computational efficiency, our implementation of the VIL does not maintain the transaction-ids, but points to the related SILs. In particular, $VIL_\alpha[j]$ points to $SIL_i[j]$ if the transaction-id i belongs to VIL_α.

Before building the CSET, the SILs of all frequent itemsets are incrementally computed. SILs for itemsets of size 1 are built during the first database scan. Then, SILs of itemsets of size greater than 1 are built during the itemset extension step (see Sect. 5.1).

In contrast, VILs are associated with frequent sequences and are built during the sequence extension step (see Sect. 5.2). Initially, for each sequence α of size 1, VIL_α is straightforwardly computed from SIL_t, where t is the only closed itemset in α. In particular, for each $j \in \{1 \ldots n\}$, $VIL_\alpha[j]$ is the first value of the list $SIL_t[j]$.

Example 6 Figure 2c shows the VIL_α of the 1-itemset sequence $\alpha = \langle\{a\}\rangle$. The value $VIL_\alpha[j]$ corresponds to the *transaction-id* of the first occurrence of the itemset $\{a\}$ in the sequence S_j, which is actually stored in $SIL_{\{a\}}[1]$ (see Fig. 2a). Therefore, $VIL_\alpha = [1, 2, 2]$.

The computation of the VILs for sequences of size greater than 1 is explained in Sect. 5.2.

5.1 I-step: using SILs

The itemset extension step (*I-Step*) is executed during the construction of the CIET. Suppose we have two sparse id-lists SIL_{i_1} (for the itemset i_1) and SIL_{i_2} (for the itemset i_2) and we want to extend the itemset i_1 with items in i_2. The SIL of $i_1 \cup i_2$ ($SIL_{i_1 \cup i_2}$) can be obtained by simultaneously scanning all the rows of SIL_{i_1} and SIL_{i_2}. In particular, for each row j, only the transaction-ids which are found in both $SIL_{i_1}[j]$ and $SIL_{i_2}[j]$ are inserted in $SIL_{i_1 \cup i_2}[j]$. Thus, $SIL_{i_1 \cup i_2}$ represents the occurrences of the itemset $i_1 \cup i_2$ in the database.

For each itemset i, its support can be efficiently computed by counting the non-null vector elements in the SIL_i. This can be done during the construction of the SIL_i at no additional cost.

Example 7 Consider the running example in Fig. 1c. Figures 2a and 1a show the sparse id-lists for itemsets $\{a\}$ and $\{b\}$. Figure 1b displays the sparse id-list for the itemset $\{a, b\}$. It contains for the first list (in row 1) only the element with value 1, for the second list the value *null*, and for the list in row 3 only the element with value 2. The support of itemset $\{a, b\}$ is 2, that is, the number of rows whose values are different from *null*.

5.2 S-step: using VILs

Consider two sibling nodes in the CSET and their corresponding sequences $\alpha = \langle a_1, a_2, \ldots, a_m \rangle$ and $\beta = \langle b_1, b_2, \ldots, b_m \rangle$. By constructing the CSET, we have that $a_m \neq b_m$ and $\forall i = 1, \ldots, m-1 : a_i = b_i$. The sequence extension step (*S-step*) of α using β aims at both constructing a new sequence $\gamma = \langle a_1, a_2, \ldots, a_m, b_m \rangle$ by appending b_m to α, and computing VIL_γ from VIL_α and VIL_β.

The computation of VIL_γ proceeds as follows. If either $VIL_\alpha[j]$ or $VIL_\beta[j]$ are *null*, i.e., α and β do not occur together in the jth sequence in *SDB*, then $VIL_\gamma[j]$ is set to *null*, since γ cannot occur in the sequence itself. If both $VIL_\alpha[j]$ and $VIL_\beta[j]$ are non-null, we have to check that an occurrence of a_m that precedes an occurrence of b_m in the jth sequence exists. Procedurally, this is performed as follows. While $VIL_\beta[j] \neq null$ &&[1] $VIL_\alpha[j] \geq VIL_\beta[j]$, the reference to $SIL_{\{b_m\}}[j]$ stored in $VIL_\beta[j]$ is used to *right-shift* to the next transaction-id in $SIL_{\{b_m\}}[j]$. At the end, if $VIL_\alpha[j] < VIL_\beta[j]$, the transaction-id found (possibly after some right-shifts) in $SIL_{\{b_m\}}[j]$ is stored in $VIL_\gamma[j]$ (check succeeded); otherwise $VIL_\gamma[j]$ is set to *null* (check failed).

During the S-Step, only closed itemsets are considered in the sequences. This guarantees a significant reduction in the search space.

Example 8 Consider the database in Fig. 1c. Let Fig. 2c, d be the VILs for the sequences $\alpha = \langle \{a\} \rangle$ and $\beta = \langle \{e\} \rangle$, respectively. Figure 2e shows the VIL of sequence $\gamma = \langle \{a\}, \{e\} \rangle$ resulting from an *S-step* on α using β.

- The initial values of $VIL_\alpha[1]$ and $VIL_\beta[1]$ are 1 and 3, respectively. Since $VIL_\alpha[1] < VIL_\beta[1]$, $VIL_\gamma[1] = 3$.
- The initial values of $VIL_\alpha[2]$ and $VIL_\beta[2]$ are 2 and 1, respectively. Since $VIL_\alpha[2] \geq VIL_\beta[2]$, the reference to $SIL_{\{e\}}[2]$ stored in $VIL_\beta[2]$ is used to identify the next transaction-id of $\{e\}$ (Fig. 2b). Since this value does not exist, $VIL_\gamma[2] = null$.
- The initial values of $VIL_\alpha[3]$ and $VIL_\beta[3]$ are 2 and 1, respectively. Since $VIL_\alpha[3] \geq VIL_\beta[3]$, the reference to $SIL_{\{e\}}[3]$ stored in $VIL_\beta[3]$ is used to identify the next transaction-id of $\{e\}$, i.e., 3. This means that $VIL_\gamma[3] = 3$.

[1] Here && denotes the shotcut AND predicate.

In the next section, the importance of both the I-step and the S-step for the CloFAST algorithm is explained.

6 CloFAST: the algorithm

In this section, we describe the CloFAST algorithm (see Algorithm 1) and the one-step technique used to simultaneously check for both sequence closure and sequence pruning.

With the first database scan, CloFAST finds the frequent 1-itemsets and builds their sparse id-lists (line 2). Then, it simultaneously discovers the closed frequent itemsets and builds their sparse id-lists (line 4). This is achieved by building a CIET, based on a modified version of the algorithm FAST [19], which integrates the marking and pruning technique proposed in Moment [8].

The first level of the CSET is initialized in lines 5–12. Each node in the first level represents a (candidate) closed sequence of size 1, whose unique element is a closed frequent itemset. The VILs of the nodes at the first level are straightforwardly computed from the SILs of the closed frequent itemsets. Starting from the first level, the nodes in the CSET are considered for sequence extension (lines 13–16) according to a depth-first search strategy.

During the mining process, the current set of closed sequential patterns is stored in the CSET. At the end, CloFAST returns the complete set of closed sequential patterns in the CSET.

Algorithm 1 CloFAST(SDB, min_sup)

Input: Sequence database SDB, int min_sup
Output: Complete set of closed freq. sequences CFS;
Data: CSET T=new Tree(), Frequent Items FI, Closed Frequent Itemset CFI, Node n;
1: // *Identify frequent 1-itemsets and build their SILs;*
2: FI = *loadFrequentSILs*(SDB, min_sup);
3: // *Identify closed frequent itemsets and their SILs*
4: CFI= *mineClosedFItemset*(FI, min_sup);
5: **for each** cfi ∈ CFI **do**
6: // *Create* VIL_{cfi} *from* SIL_{cfi}
7: vil=*createVil*(cfi);
8: // *Create CSET node associated to cfi*
9: n= *createNode*(cfi,vil);
10: *labelNodeAs*(n,"closed");
11: *addChildNode*(T,root(T),n);
12: **end for**
13: **for each** child ∈ *children*(T,root(T)) **do**
14: // *s*tart the depth first search
15: *sequenceExtension*(T,child,min_sup);
16: **end for**
17: **return** *closedSequentialPatterns*(T);

Algorithm 2 describes the sequence extension step for an input node n. It first tests whether n is closed and/or can be pruned (line 1). This is achieved by means of the `checkClosureAndPrune` method detailed in the next subsections. If n is not pruned (line 2), for each of its siblings including itself, the *S-step* is executed, in order to generate its children sequences (lines 6–20). Only the new frequent sequences are stored in the CSET, together with their corresponding VILs (denoted as $v3$ in the algorithm). If a generated sequence has the same support as that represented in n, then n is labeled as *non-closed* (lines

11–13); otherwise it is labeled as *closed* by default (it is indeed a candidate closed frequent sequence). In lines 21–23, the sequence extension step is recursively applied to each child of n.

Algorithm 2 SequenceExtension(T, n, min_sup)

```
Input: CSET T, Node n, int min_sup;
Data: Node u, newNode, child; VIL v1,v2,v3; Sequence newSequence; int supp
 1: checkClosureAndPrune(n, T);
 2: if pruned(n); then
 3:   return
 4: end if
 5: v1 = Vil(n);
 6: for each u ∈ siblings(T,n) do
 7:   v2 = Vil(u);
 8:   (v3,supp) = S-Step(v1,v2); // create the new vertical id-list and compute its support
 9:   if supp ≥ min_sup then
10:     if supp = support(v1); then
11:       labelNodeAs(n,"nonClosed");
12:     end if
13:     // create new CSET node
14:     newSequence =extend(sequence(n), sequence(u));
15:     newNode = createNode(newSequence, v3);
16:     labelNodeAs(newNode,"closed");
17:     addChildNode(T, n, newNode);
18:   end if
19: end for
20: for each child ∈ children(T, n); do
21:   sequenceExtension(T, child, min_sup);
22: end for
```

6.1 Backward closure checking

Inspired by BIDE [22], we aim at pruning the search space by exploiting a closure checking schema which, besides the forward construction of the CSET, operates in a backward fashion. Closure checking is important since it is useless to further explore a node if this node, and its descendants could be absorbed by nodes present in other paths of the tree.

The intuition behind the backward solution is that it would lead to first check sequences in the tree which are more "similar" to the sequence α to be evaluated (same head, same length, closer in the tree). This means that, if a sequence which absorbs α exists, the backward closure checking is faster than a classical top-down solution. If there is no such sequence, the two approaches are equivalent.

Methods for pruning frequent closed itemsets have already been presented in the literature [8]. However, search space pruning in closed frequent sequence mining is trickier than in closed frequent itemset mining. Indeed, while a depth-first-search-based closed itemset mining algorithm can *safely* stop growing a prefix itemset as soon as it finds that this itemset can be absorbed by another closed itemset already generated, a closed sequence mining algorithm needs additional checks. This is due to both the possible presence of multiple instances of the same itemset in a sequence and to the ordering among the itemsets in the sequence.

Pruning is rather complex in BIDE, since it is based on pseudo-projected databases. On the contrary, CloFAST takes advantage of the VIL data structures, which convey essential information for pruning. Indeed, it is useless to expand a node n if, in other branches of the tree

a node exists that has the same VIL as n and represents a sequence which is a supersequence of that represented in n.

This is described in Algorithm 3. Given a CSET node n representing a sequence $\alpha = \langle a_1, a_2, \ldots, a_m \rangle$, `checkClosureAndPrune` first checks whether α is closed. If not, it checks whether n can be safely pruned. As defined in Sect. 2, α is non-closed if any supersequence β of α exists that absorbs α. This definition can be used for *backward closure*.

Algorithm 3 checkClosureAndPrune(n, T)

```
Input: Node n, CSET T;
Data: Node u; List siblings ; VIL vilU, vilP;
 1: i = level(n);
 2: n'= parent(T,n);
 3: repeat
 4:   vilP = Vil(n');
 5:   children = children(T, n');
 6:   for each u ∈ children do
 7:     vilU = Vil(u);
 8:     // check if last itemset of u contains the i-th itemset of n
 9:     if contains(lastItemset(u), itemset(n,i)) then
10:       if itemsetClosure(vilU, path(n', n)) then
11:         labelNodeAs(n,"nonClosed");
12:         if earlyTermination(vilU,vilP) then
13:           labelNodeAs(n,"pruned");
14:         end if
15:         return;
16:       end if
17:     end if
18:     if sequenceClosure(vilU, path(n', n)) then
19:       labelNodeAs(n,"nonClosed");
20:       if earlyTermination(vilU,vilP) then
21:         labelNodeAs(n,"pruned");
22:       end if
23:       return;
24:     end if
25:   end for
26:   i= i-1;
27:   n' = parent(T,n');
28: until n' ≠ root(T);
```

Definition 3 (*Backward Closure*) Let $\alpha = \langle a_1, a_2, \ldots, a_m \rangle$ be a frequent sequence of size m. Then, α is non-closed in *backward closure* if for some $i \in \{1 \ldots m\}$ an itemset b exists, such that one of the following two conditions holds:

1. $a_i \subset b$ and $\langle a_1, \ldots, a_{i-1}, b, a_{i+1}, \ldots, a_m \rangle$ has the same support as α (*itemset closure*);
2. $\langle a_1, \ldots, a_{i-1}, b, a_i, \ldots, a_m \rangle$ has the same support as α (*sequence closure*).

Given the sequence α represented in the node n, Algorithm 3 identifies an itemset b which allows us to check whether α is non-closed in backward closure or not. The search can be safely restricted to the last itemset of the sequences represented in the siblings of either n or its ancestors. This is done by using only the CSET, which is climbed level-by-level, starting from the direct parent n' of n (line 2) and considering each child u of n' (line 6). The algorithm identifies candidate sequences in the form of $\gamma = \langle a_1, \ldots, a_{i-1}, b \rangle$, represented in u. Such candidate sequences are then used in order to evaluate the support of either

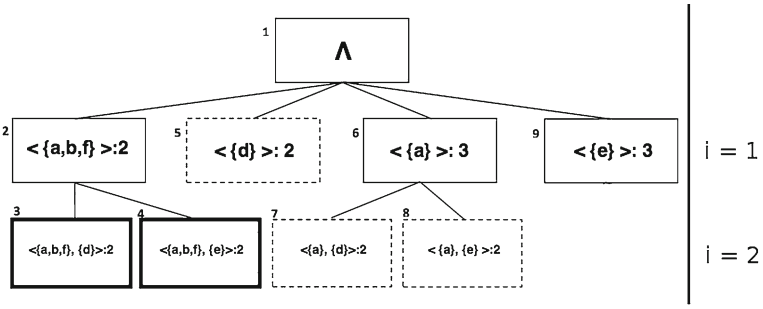
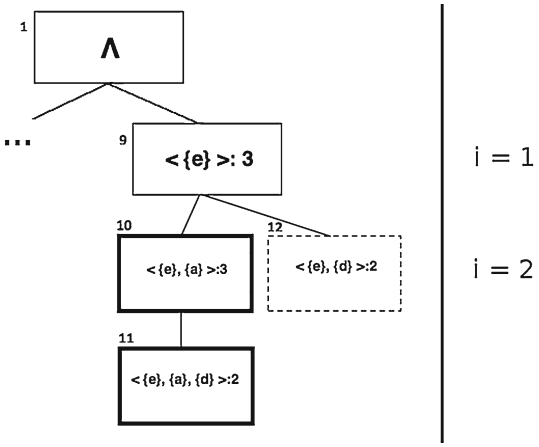


Fig. 5 Partial view of the CSET for the dataset in Fig. 1c

Fig. 6 Partial view of the CSET for the dataset in Fig. 1c



$\beta = \langle a_1, \ldots, a_{i-1}, b, a_{i+1}, \ldots, a_m \rangle$, if $a_i \subset b$, or $\beta = \langle a_1, \ldots, a_{i-1}, b, a_i, \ldots, a_m \rangle$, in any case. It is noteworthy that, during the identification of candidate sequences in the form of γ, neither is the CSET modified nor are new sequences evaluated. Moreover, the computation of the support of the sequences β only exploits VILs, as we will explain later.

Examples 9 and 10 clarify these aspects for the two cases in Definition 3.

Example 9 (Itemset closure) Consider the dataset in Fig. 1c and the sequence $\alpha = \langle \{a\}, \{d\} \rangle$ represented in node 7 of Fig. 5. CloFAST examines at the first step (level $i = 2$) the sequence $\gamma = \langle \{a\}, \{e\} \rangle$ represented in node 8 (sibling of 7). The last itemset of γ (i.e., $\{e\}$) does not contain the last itemset of α (i.e., $\{d\}$), so the itemset closure cannot be checked. CloFAST moves at the previous level ($i = 1$) and checks the itemset closure of α over the itemset $\{a\}$ using the children of the CSET's root (nodes 2, 5, 6, 9). Given $\gamma = \langle \{a, b, f\} \rangle$ (node 2), since the last itemset of γ contains the last but one itemset of α (i.e., $\{a\}$), CloFAST checks whether the supersequence $\beta = \langle \{a, b, f\}, \{d\} \rangle$ can absorb α. In that case, α is labeled as *non-closed*.

Example 10 (Sequence closure) Consider the dataset in Fig. 1c and the sequence $\alpha = \langle \{e\}, \{d\} \rangle$ (see node 12 in Fig. 6). CloFAST examines at the first step (level i=2) the children of the parent of α, i.e., the sequence $\gamma = \langle \{e\}, \{a\} \rangle$ (node 10). By inserting $\{a\}$, the last itemset of sequence γ, between the itemsets $\{e\}$ and $\{d\}$ of α, we obtain the sequence $\beta = \langle \{e\}, \{a\}, \{d\} \rangle$ (node 11), which absorbs α. Thus, α is labeled as *non-closed*.

As previously mentioned, backward closure is verified by only working on the CSET and VILs. Algorithmically, the predicate `contains` (line 9) checks whether the ith itemset of α is a proper subset of the last itemset in γ. In this case, the predicate `itemsetClosure` is executed to check whether β absorbs α. If the `itemsetClosure` is false, CloFAST checks the `sequenceClosure` predicate (line 18).

In order to explain how backward closure is verified in CloFAST, we define two predicates, namely *shiftSC* (shift sequence closure) and *shiftIC* (shift itemset closure). The former (latter) works on the VILs and SILs to check whether whenever the jth sequence in *SDB* contains α, it also contains the supersequence β constructed as in Definition 3—case 2 (case 1). Obviously, the opposite is always true, i.e., whenever the jth sequence in *SDB* contains the supersequence β, it also contains the subsequence α. Therefore, if either *shiftSC* or *shiftIC* hold for each j, then α and β have the same support, i.e., β absorbs α (or α is non-closed).

Definition 4 (*shiftSC*) Let $\alpha = \langle a_1, \ldots, a_{i-1}, a_i, \ldots a_m \rangle$ be the frequent sequence for which we intend to verify `sequenceClosure` (at the ith level), $\delta = \langle a_1, \ldots, a_{i-1}, a_i \rangle$ be the $(m-i)$th ancestor of α and $\gamma = \langle a_1, \ldots, a_{i-1}, b \rangle$ be a sibling of δ. Let the jth sequence in *SDB* contain α, i.e., $VIL_\alpha[j] \neq null$. Then, the predicate *ShiftSC*, which takes as input both $VIL_\gamma[j]$ and the list of the VILs stored in the path from δ to α,[2] is recursively defined as follows:

$$shiftSC(VIL_\gamma[j], [VIL_\delta[j], VIL_{\langle a_1,\ldots,a_{i+1}\rangle}[j], VIL_{\langle a_1,\ldots,a_{i+2}\rangle}[j], \ldots, VIL_\alpha[j]])$$
$$= \begin{cases} true \ \text{ if } \ \left(\begin{array}{l} VIL_\gamma[j] < VIL_\delta[j] \\ \vee \ \exists t_{a_i} \in SIL_{a_i}[j], t_{a_i} \neq null \text{ such that} \\ \qquad \left(VIL_\gamma[j] < t_{a_i} \wedge shiftSC(t_{a_i}, [VIL_{\langle a_1,\ldots,a_{i+1}\rangle}[j], \ldots, VIL_\alpha[j]])\right) \end{array} \right) \\ false \text{ otherwise} \end{cases}$$

Thus, *shiftSC* checks whether $VIL_\gamma[j] < VIL_\delta[j]$, i.e., b, the last itemset of γ, can precede a_i, the last itemset of δ in the jth sequence. If so, from proposition 5 the jth sequence in *SDB* contains the supersequence $\beta = \langle a_1, \ldots, a_{i-1}, b, a_i, \ldots a_m \rangle$. If not, the check is repeated on a virtual shift to the next transaction-id in the list $SIL_{a_i}[j]$. This amounts to determining an alternative value, if any, for $VIL_\delta[j]$, such that $VIL_\beta[j] \neq null$, i.e., the sequence $\langle a_{i+1}, \ldots a_m \rangle$ can be juxtaposed to $\langle a_1, \ldots, a_{i-1}, b, a_i \rangle$ by preserving the containment relationship for the jth sequence.

If the conditions stated in Definition 4 are satisfied for all non-null values of VIL_α, then α is labeled as *non-closed*.

Definition 5 (*shiftIC*) Let $\alpha = \langle a_1, \ldots, a_{i-1}, a_i, \ldots a_m \rangle$ be the frequent sequence for which we intend to verify the `itemsetClosure` (at the ith level), $\delta = \langle a_1, \ldots, a_{i-1}, a_i \rangle$ be the $(m-i)$th ancestor of α and $\gamma = \langle a_1, \ldots, a_{i-1}, b \rangle$ be a sibling of δ, such that $a_i \subset b$. Then, the predicate *ShiftIC*, which takes as input $VIL_\gamma[j]$ and the list of the VILs stored in the path from δ to α, is defined as follows:

$$shiftIC(VIL_\gamma[j], [VIL_\delta[j], VIL_{\langle a_1,\ldots,a_{i+1}\rangle}[j], VIL_{\langle a_1,\ldots,a_{i+2}\rangle}[j], \ldots, VIL_\alpha[j]])$$
$$= \begin{cases} true \ \text{ if } \ \left(\begin{array}{l} VIL_\gamma[j] = VIL_\delta[j] \\ \vee \ \exists t_{a_i} \in SIL_{a_i}[j], t_{a_i} \neq null \text{ such that} \\ \qquad \left(t_{a_i} = VIL_\gamma[j] \wedge shiftSC(t_{a_i}, [VIL_{\langle a_1,\ldots,a_{i+1}\rangle}[j], \ldots, VIL_\alpha[j]])\right) \end{array} \right) \\ false \text{ otherwise} \end{cases}$$

[2] In order to simplify the notation, we will use the term sequence to identify the CSET node that contains the sequence itself.

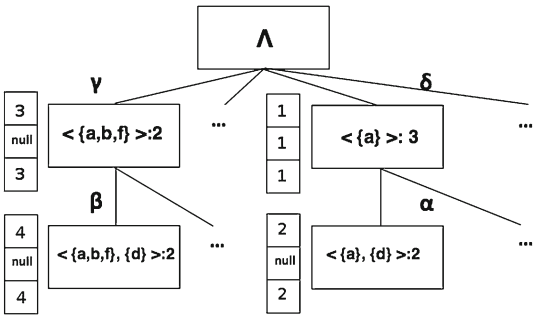


Fig. 7 Example of itemset closure for sequence $\alpha = \langle\{a\}, \{d\}\rangle$. On the *left* of each node the corresponding VIL is shown

Thus, *shiftIC* checks whether $VIL_\gamma[j] = VIL_\delta[j]$, i.e., the last itemset of δ can be replaced by the last itemset of γ. If so, from proposition 6 the jth sequence in *SDB* contains the supersequence $\beta = \langle a_1, \ldots, a_{i-1}, b, a_{i+1}, \ldots a_m\rangle$. If not, the check is repeated on a virtual shift to the next transaction-id in the list $SIL_{a_i}[j]$, which amounts to determining an alternative value, if any, for $VIL_\delta[j]$ such that $VIL_\beta[j] \neq null$.

It is noteworthy that the definition of *shiftIC* depends on *shiftSC*, since we need to check that the rest of the sequence $\langle a_{i+1}, \ldots a_m\rangle$ can be juxtaposed to $\langle a_1, \ldots, a_{i-1}, b\rangle$ by preserving the containment relationship for the jth sequence. If the conditions stated in Definition 5 are satisfied for all non-null values of VIL_α, then α is labeled as *non-closed*.

Since *shiftSC* and *shiftIC* coincide, apart from the test on the VILs ($<$ for shiftSC and $=$ for shiftIC), we show an example only for the *shiftIC* predicate.

Example 11 Consider the following database *SDB*:

1. $\langle\{a\}, \{d\}, \{a, b, f\}, \{d\}\rangle$,
2. $\langle\{a\}, \{c\}\rangle$,
3. $\langle\{a\}, \{d\}, \{a, b, f\}, \{d\}\rangle$,

and the sequence $\alpha = \langle\{a\}, \{d\}\rangle$. A partial view of the corresponding CSET is reported in Fig. 7. According to the definition of `itemsetClosure`, α is non-closed if, for each $j \in [1, \ldots, n]$ such that $VIL_\alpha[j] \neq null$, the predicate *shiftIC* is true. Since at level 2 (last level) there is no child whose last itemset can replace the last itemset of α, CloFAST moves up to the previous level and analyzes the children of the root. At this level, CloFAST checks whether the first itemset of α, i.e., $\{a\}$, can be replaced by the last itemset of the sequence γ, i.e., $\{a, b, f\}$. Consider the first sequence, i.e., $j = 1$. Since $VIL_\delta[1] = 1$ differs from $VIL_\gamma[1] = 3$, CloFAST checks whether it is possible to shift to the next transaction-id in the list $SIL_{\{a\}}[1]$. This leads to a virtual "shifting" of transaction-ids of $VIL_\delta[j]$ until $VIL_\delta[1] = VIL_\gamma[1]$ is satisfied. This is true since $SIL_{\{a\}}[1] = [1, 3]$. As a second step, CloFAST checks that the "new" value of $VIL_\delta[1]$, i.e., 3, is less than $VIL_\alpha[1]$, i.e., 2. Since this check fails, CloFAST performs a virtual "shifting" of $VIL_\alpha[1]$ using $SIL_d[1] = [2, 4]$ and obtains a "new" value of $VIL_\alpha[1]$, namely 4, such that $VIL_\delta[1] < VIL_\alpha[1]$ $(3 < 4)$. Since for each j such that $VIL_\alpha[j] \neq null$, i.e., j=1,3, the predicate *shiftIC* holds, α is labeled as *non-closed*.

The theoretical motivation for itemset closure checking originates from the following theorem.

Theorem 1 (`itemsetClosure`) *Let*

– *SDB be a sequence database,*

- $\alpha = \langle a_1, \ldots, a_{i-1}, a_i, \ldots a_m \rangle$ *be the frequent sequence in SDB to be checked for itemset closure on the* i*th itemset* a_i*,*
- $\delta = \langle a_1, \ldots, a_{i-1}, a_i \rangle$ *be the* $(m-i)$*th ancestor of* α *in the CSET,*
- $\gamma = \langle a_1, \ldots, a_{i-1}, b \rangle$ *be a sibling of* δ *such that* $a_i \subset b$*, and*
- $\beta = \langle a_1, \ldots, a_{i-1}, b, a_{i+1}, \ldots a_m \rangle$ *be a supersequence of* α*.*

If:

$$\forall j = 1, \ldots, n : (VIL_\alpha[j] \neq null \Rightarrow shiftIC(VIL_\gamma[j], [VIL_\delta[j], \ldots, VIL_\alpha[j]])) \quad (1)$$

then β *absorbs* α*.*

Proof By definition of *shiftIC*, if $VIL_\alpha[j] \neq null$ and $shiftIC(VIL_\gamma[j], [VIL_\delta[j], \ldots, VIL_\alpha[j]])) = true$, then $VIL_\beta[j] \neq null$.

Thus, sequences that contain α also contain β. Vice versa, since β is a supersequence of α, sequences that contain β also contain α. This means that the support of α is the same as the support of β, i.e., β absorbs α. □

Theorem 2 provides sufficient conditions for sequence closure checking.

Theorem 2 (`sequenceClosure`) *Let*

- *SDB be a sequence database,*
- $\alpha = \langle a_1, \ldots, a_{i-1}, a_i, \ldots a_m \rangle$ *be the frequent sequence in SDB to be checked for sequence closure on the* i*th itemset* a_i*,*
- $\delta = \langle a_1, \ldots, a_i \rangle$ *be the* $(m-i)$*th ancestor of* α*,*
- $\gamma = \langle a_1, \ldots, a_{i-1}, b \rangle$ *be a sibling of* δ *and*
- $\beta = \langle a_1, \ldots, a_{i-1}, b, a_i, \ldots a_m \rangle$ *be a supersequence of* α*.*

If:

$$\forall j = 1, \ldots, n : (VIL_\alpha[j] \neq null \Rightarrow shiftSC(VIL_\gamma[j], [VIL_\delta[j], \ldots, VIL_\alpha[j]])) \quad (2)$$

then β *absorbs* α*.*

Proof The proof is analogous to that of Theorem 1. □

6.2 Pruning

If a node n is labeled as *non-closed*, then it is evaluated for pruning (Algorithm 3, lines 12–13 and 21–22). Indeed, it is possible that a *non-closed* sequence can still be profitably used for generating closed sequences. As described in Example 11, the sequence $\alpha = \langle \{a\}, \{d\} \rangle$ is *non-closed* because $\beta = \langle \{a, b, f\}, \{d\} \rangle$ absorbs α. However, it can be used to generate, in sequence extension, the closed sequence $\langle \{a\}, \{d\}, \{a, b, f\} \rangle$, which cannot be generated if the subtree rooted in the node associated with the sequence α is pruned.

On the other hand, there are cases in which *non-closed* sequences cannot lead to the generation of closed sequences. In these cases, their corresponding nodes should be labeled as pruned, in order to prevent CloFAST from generating further unpromising patterns. The following proposition provides the theoretical basis for pruning.

Proposition 7 *Let* $\alpha = \langle a_1, a_2, \ldots, a_m \rangle$ *be a frequent sequence,* $\beta = \langle b_1, b_2, \ldots, b_p \rangle$ *a supersequence of* α *and* N *the number of the elements in the* VIL_α *which differ from the corresponding transaction-ids in the* VIL_β*. If* $N = 0$*, i.e.,* $VIL_\alpha = VIL_\beta$*, then for the two sequence extensions* $\gamma = \langle a_1, a_2, \ldots, a_m, c_1, c_2, \ldots, c_q \rangle$ *and* $\delta = \langle b_1, b_2, \ldots, b_p, c_1, c_2, \ldots, c_q \rangle$*,* $VIL_\gamma = VIL_\delta$*.*

Proof By induction on q.

- *Base case*: $q = 0$. Trivial.
- *Induction step*: $q > 0$. Consider the two sequences $\alpha' = \langle a_1, a_2, \ldots, a_m, c_1, c_2, \ldots, c_{q-1}\rangle$ and $\beta' = \langle b_1, b_2, \ldots, b_p, c_1, c_2, \ldots, c_{q-1}\rangle$. By construction, β' is a supersequence of α'. If $N = 0$, by inductive hypothesis, $VIL_{\alpha'} = VIL_{\beta'}$. If we juxtapose the same itemset c_q to both sequences, the VILs of the two extended sequences $\gamma = \langle a_1, a_2, \ldots, a_m, c_1, c_2, \ldots, c_q\rangle$ and $\delta = \langle b_1, b_2, \ldots, b_p, c_1, c_2, \ldots, c_q\rangle$, will still be the same, i.e., VIL_γ=VIL_δ.

□

It is noteworthy that γ and δ have the same support. Since δ is a supersequence of γ, then δ absorbs γ. Therefore, it is useless to generate the extensions of α, since they will be absorbed by the sequences generated from β (*early termination condition*).

In CloFAST, this early termination condition is efficiently checked during both the *itemset closure* and *sequence closure* phases (lines 12, 20) at no additional cost. In particular, if for each j such that $VIL_\alpha[j] \neq null$, the predicates *shiftIC* or *shiftSC* are satisfied without applying the virtual "shifting," then the node representing α can be labeled as pruned.

Example 12 Consider the *itemset closure* described in Example 9 and depicted in Fig. 5. At the first level ($i = 1$), CloFAST applies the *ShiftIC* predicate having as arguments the VIL of node 2 ($\gamma = \langle\{a, b, f\}\rangle$) and the VILs of the path between nodes 6 ($\delta = \langle\{a\}\rangle$) and 8 ($\alpha = \langle\{a\}, \{d\}\rangle$). Since, for each j such that $VIL_\alpha[j] \neq null$ (i.e., $j = 1$ and $j = 3$), $VIL_\gamma[j] = VIL_\delta[j]$ (in particular, $VIL_\gamma = VIL_\delta = [1, 2, 2]$), then the sequence $\beta = \langle\{a, b, f\}, d\rangle$ exists that has the same VIL as α and will generate the same supersequences as α. For this reason, node 2, which represents the sequence α, can be labeled as pruned.

Recalling that the backward closure is verified by only working on VILs, the time complexity of Algorithm 3 is $O(d \cdot |SDB|)$, where d is the depth of the tree. Therefore, the backward closure can be efficiently checked in practical cases characterized by relatively small values of d.

7 Experiments

In order to empirically evaluate CloFAST, in this section we report the experimental results on both real-world and synthetic datasets. We implemented CloFAST in Java and compared it with BIDE, ClaSP and CloSpan provided by the Java framework SPMF [10].[3] The experimental setting is inspired by [22] and aims at evaluating:

- *Efficiency* Efficiency is evaluated both in terms of running time (seconds) and memory consumption (Gb) on sparse and dense datasets. Following [12], we define the density as the ratio between the average number of items in an itemset and the number of different items. When this value is small, the generated dataset is considered sparse, whereas when this ratio is high, the dataset is considered dense. We compare CloFAST efficiency both on synthetic and real datasets.
- *Scalability* CloFAST is compared with the above-cited algorithms by linearly increasing the number of input sequences. We report the results in terms of running time (seconds)

[3] Unfortunately, we were not able to compare CloFAST with COBRA, which is not publicly available. Moreover, the algorithm description provided in [14] does not provide enough details to unambiguously re-implement the system.

Table 2 Parameters used in the IBM data generator

Parameter	Description
D	Number of sequences ($*10^3$)
C	Average number of itemsets per sequence
T	Average number of items per itemset
S	Average length of maximal sequences
I	Average size of itemsets in maximal sequences
N	Number of different items ($*10^3$)

In the definition of S and I, a sequence is considered maximal if it is not a subsequence of any other frequent sequence [25]

and memory consumption (MB). Scalability is only evaluated on artificially generated datasets.

– *Effectiveness of the CloFAST optimization technique* CloFAST is compared with FAST which does not implement the *backward closure checking* and *pruning* techniques. We report results in term of running time (seconds), memory consumption (GB) and number of mined frequent patterns. This comparison is performed on real datasets.

All the results reported in this section are obtained with a machine with a 4-core 2.4GHZ Intel Xeon processor, running Ubuntu 12.04 Server edition with 32GB of main memory. In order to facilitate the replication of the experiments, the system and all the considered datasets can be downloaded at the following hyperlink: http://www.di.uniba.it/~ceci/micFiles/systems/CloFAST/.

Before presenting the results obtained, we describe the datasets used in the experiments.

7.1 Dataset description

The synthetic datasets used for our experiments were obtained using the IBM data generator [1]. This dataset generator has been used in most sequential pattern mining studies [1,12,18,25]. Generated datasets contain random sequences of itemsets which can be easily controlled by the user. In particular, the generator allows the user to specify several parameters which regulate, among other aspects, the number of sequences, the average number of transactions per sequence and the number of different items. The detailed list of parameters used in this evaluation is listed and explained in Table 2. The parameter values are reported in the following subsections and depend on the specific purpose of each empirical evaluation.

We also compared the algorithms on real datasets, that is, Gazelle, Snake, MSNBC and Pumsb. For all the datasets, except Snake, we also considered variants which are commonly used in the literature. We indicate such variants with the star (*) suffix. The properties of all the real datasets used in our experiments are reported in Table 3 and described in the following:

– Gazelle (BMS-WebView-1) is a dataset used in the KDDCup-2000 competition and, basically, it includes a set of page views done by users on the gazzelle.com e-commerce web site. Product pages viewed in one session are considered an itemset, and different sessions for one user define the sequence. Gazelle* represents another version of the dataset proposed in the KDDCup-2000 competition and used in past studies on sequential pattern mining [22]. Both datasets are considered sparse datasets. Gazzelle was downloaded from www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php, while Gazzelle* was downloaded from the KDD Cup 2000 Web site.
– MSNBC is a dataset of click-stream data (from the UCI repository). They are collected from logs of www.msnbc.com and news-related portions of www.msn.com for the entire day of September 28, 1999. Each sequence in the dataset corresponds to page views

Table 3 Properties of the real datasets considered for the experiments

Dataset	#Seq.	Avg length	Max length	#Items	Density
Gazelle	59,601	2.51	267	497	0.002
MSNBC	989,818	4.70	14,795	17	0.06
Pumsb	49,046	50.48	63	2088	0.0005
Gazelle*	29,369	2.98	651	1423	0.0007
Snake*	163	6.62	61	21	0.04
MSNBC*	31,790	13.33	100	17	0.06
Pumsb*	9230	50.49	61	1676	0.0006

of a user during that 24-h period. Each transaction in the sequence corresponds to a user's request for a page. MSNBC was downloaded from http://archive.ics.uci.edu/ml/datasets/MSNBC.com+Anonymous+Web+Data, while MSNBC* has been downloaded from www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php.

– Snake is a biological dataset which contains 192 Toxin-Snake protein sequences and 20 unique items. This Toxin-Snake dataset is about a family of eukaryotic and viral DNA binding proteins and was used in [22]. For our experiments, only sequences containing more than 50 items were kept. This filtering is performed in order to make the dataset more uniform (because the original Snake dataset contains only a few very short sequences and many long sequences). The dataset obtained (called Snake*) contains 163 long sequences with an average of 60.62 items. This dataset is not publicly available.
– Pumsb contains census data for population and housing from PUMS (Public Use Microdata Sample) [3]. Both Pumsb and Pumsb* were downloaded from http://fimi.ua.ac.be/data/.

7.2 Results: efficiency of CloFAST on synthetic datasets

As previously stated, to test the efficiency of CloFAST, we adopted the schema based on sparse and dense datasets proposed by Gomariz et al. [12]. They showed how the performance of the sequential pattern mining algorithms largely depends on the database density, and they introduced a definition of density based on T/N (see Table 2). When T/N is small, the generated dataset is sparse, while when T/N grows, the dataset tends to be dense.

To evaluate and compare the efficiency of the algorithms, we considered four configurations. In the first, we fixed $D = 5$ (number of transactions $*10^3$), $C = 10$ (the sequence length), $T = 10$ (number of items in an itemset) and varied N (the number of different items). We obtained the datasets D5C10T10N2.5S6I4, D5C10T10N1.6S6I4 and D5C10T10N1S6I4. In the second, we fixed $D = 50$, $C = 20$, $N = 2.5$ and varied T, obtaining the datasets D50C20T10N2.5S6I4, D50C20T20N2.5S6I4, D50C20T30N2.5S6I4 and D50C20T40N2.5S6I4, which are denser than the datasets belonging to the first configuration.

In Fig. 8, we compare CloFAST with ClaSP, BIDE and CloSpan in terms of the running time (in seconds) and memory consumption (in GB), according to the first dataset configuration and varying the support threshold. In terms of running time (graphics are reported in logarithmic scale), CloFAST generally outperforms all the other systems, especially for low support values, when the number of frequent sequences is higher. By increasing the

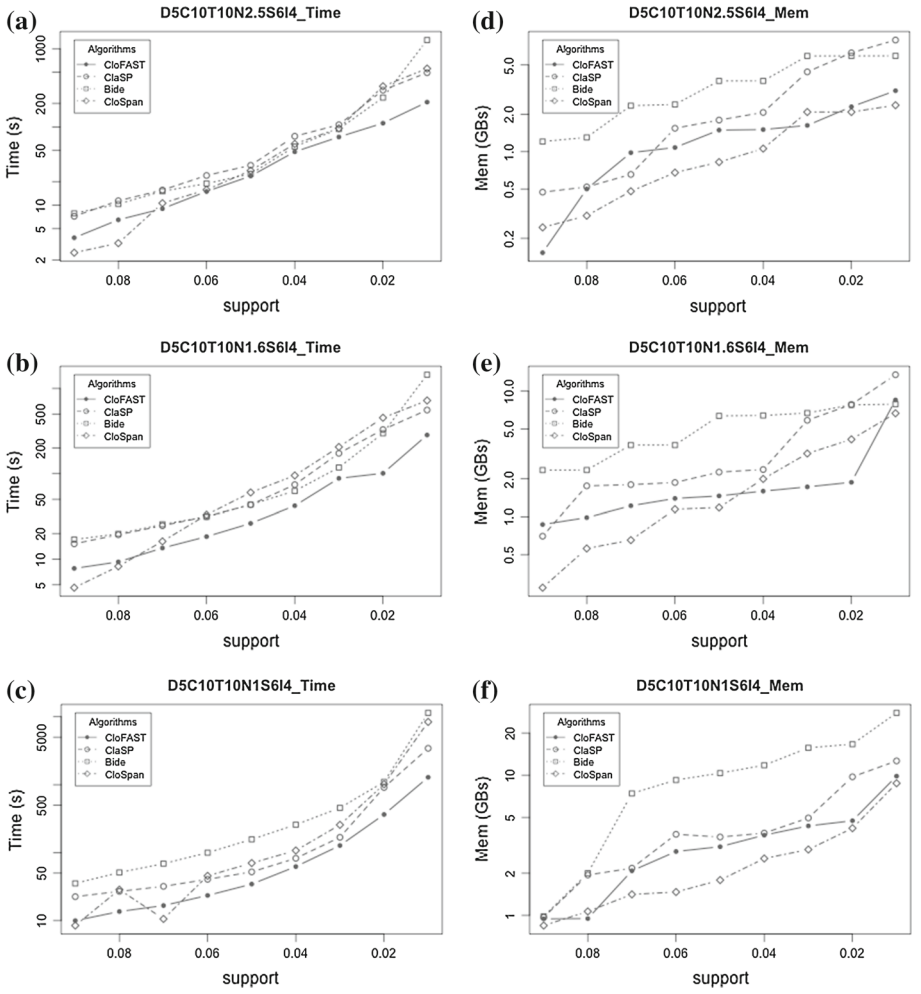


Fig. 8 Running times (in seconds) and memory consumption (in GB) varying $N = \{2.5, 1.6, 1\}$ and min_sup. Results are obtained with $D = 5$, $C = 10$ and $T = 10$. **a** D5C10T10N2.5S6I4, **b** D5C10T10N1.6S6I4, **c** D5C10T10N1S6I4, **d** D5C10T10N2.5S6I4, **e** D5C10T10N1.6S6I4, **f** D5C10T10N1S6I4

density of the dataset (i.e., by decreasing N), the advantage of CloFAST over the other three algorithms becomes more evident. Since the higher density is directly related to the number of frequent sequences, we can conclude that the higher the number of frequent sequences, the more competitive (in running time) the proposed algorithm. Notably, the time efficiency of CloFAST is not obtained at the cost of higher memory consumption, which remains comparable to that of CloSpan. For highly dense datasets and for small values of the support threshold, the worst performing system is BIDE. This is probably related to the fact that, for dense datasets, the size of the projected databases does not shrink during the mining process. The situation is more favorable to BIDE for very sparse datasets and for small values of the support threshold, thus confirming the conclusions reported in [22].

In Fig. 9, we show the results obtained according to the second dataset configuration (i.e., by varying T) and setting the support threshold to 0.4. They confirm the discussion reported

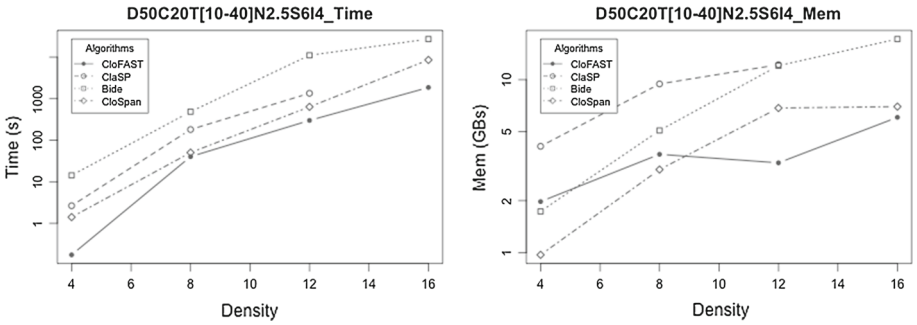


Fig. 9 Running times (in seconds) and memory consumption (in GB) varying $T/N = \{4, 8, 12, 16\}$. Results are obtained with min_sup $= 0.4$, $D = 50$, $C = 20$, $N = 2.5$

for Fig. 8, particularly that CloFAST outperforms the algorithms BIDE, ClaSP and CloSpan when the density of the datasets increases. It is noteworthy that ClaSP does not return results with the dataset D50C20T40N2.5S6I4 ($T/N = 16$), since it consumes all the assigned memory (fixed to 32GB).

Moreover, the efficiency of CloFAST with distinct density values is evaluated by varying the number of itemsets in the sequences (C). In Fig. 10, we show the running time and memory consumption of the considered algorithms using a third and a fourth dataset configuration. For the sparsest configuration ($T = 2.5$, $N = 10$, $D = 20$), we compare the performances obtained with four datasets (D20C20T2.5N10S6I4, D20C40T2.5N10S6I4, D20C60T2.5N10S6I4, D20C80T2.5N10S6I4) and 2 support thresholds. For the densest configuration ($T = 20$, $N = 4$, $D = 10$), we obtained the datasets D10C20T20N5S6I4, D10C40T20N5S6I4, D10C60T20N5S6I4, D10C80T20N5S6I4 and showed the results only for one support threshold. We observe that for the densest configuration, it was not possible to test lower support thresholds, due to the extremely large number of frequent sequences. The results show that, in general, by increasing the number of itemsets in the sequences (C), CloFAST shows lower running times than other systems. This behavior is more evident for the more complex task of mining dense datasets with a high number of itemsets in the sequences (and a high number of frequent patterns). In this case, CloFAST outperforms competitors by one order of magnitude (see Fig. 10c), while keeping memory consumption under control (see Fig. 10f). Concerning this last aspect, we observe again a good behavior of CloSpan in terms of memory consumption. This effect is explained by the efficient way CloSpan stores internal data structures (integer vectors), which allows it to save memory at the price of higher running times (note that running times are expressed in logarithmic scale, while memory consumption is expressed in linear scale).

Finally, we selected one experiment from the first, the second and the fourth configuration (median of values of other parameters) and varied S and I, obtaining the datasets D5C10T20N1.6S[2..10]I[2..10], D50C20T20N2.5S[2..10]I[2..10] and D10C60T20N5S[2..10]I[2..10]. In this way, it was possible to evaluate how the parameters S and I affected the computation time on the selected datasets. In Figs. 11 and 12, we report the results obtained. From the twelve heatmaps, we can conclude that CloFast has the same trend as other algorithms but, coherently with the results reported before, it is the best performing in the case of dense datasets. In particular, on dense datasets, CloFast outperforms competitors by a good margin when the values of I and S are small (top-left corner of the heatmap), i.e., when the number of frequent patterns is higher.

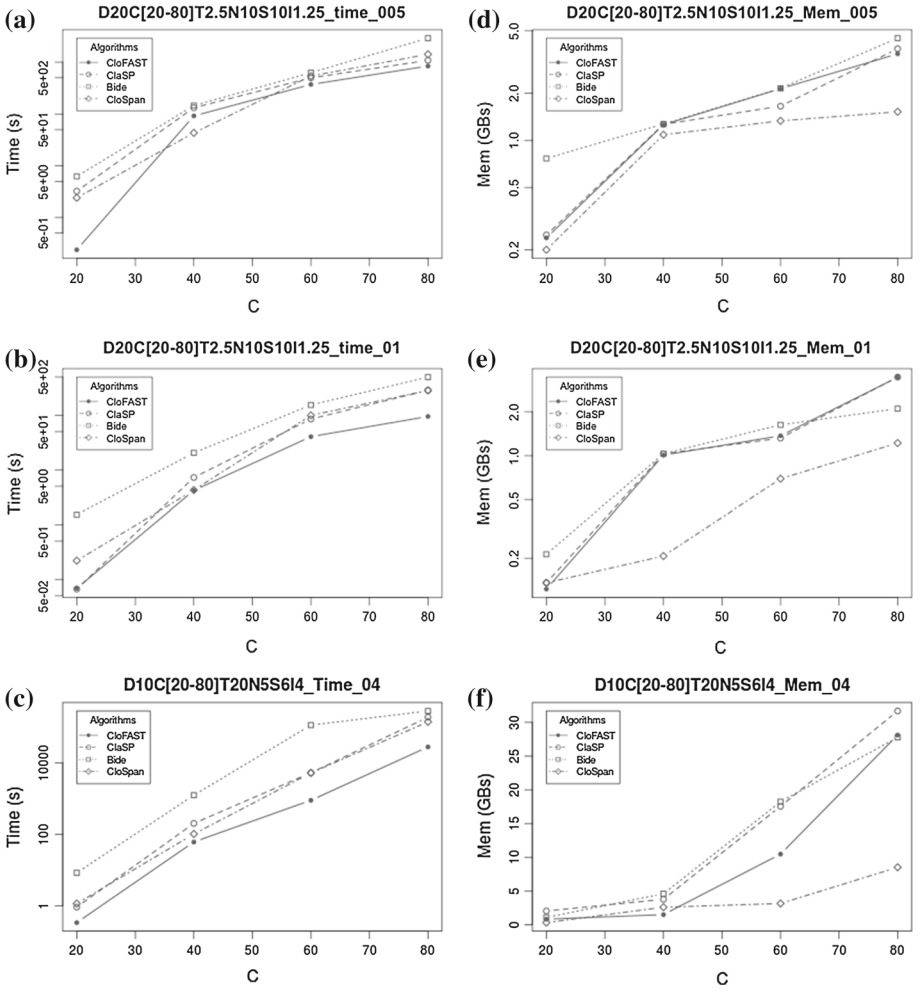


Fig. 10 Running times (in seconds) and memory consumption (in GB) varying $C = \{20, 40, 60, 80\}$. Results are obtained with $D = 20$, min_sup $= 0.05$, $T = 2.5$ (sparse); $D = 20$, min_sup $= 0.1$, $T = 2.5$ (sparse); $D = 10$, min_sup $= 0.4$, $T = 20$ (dense). **a** D20C[20-80]T2.5N10S10I1.25 min_supp $= 0.05$, **b** D20C[20-80]T2.5N10S10I1.25 min_supp $= 0.1$, **c** D10C[20-80]T20N5S6I4 min_supp $= 0.4$, **d** D20C[20-80]T2.5N10S10I1.25 min_supp $= 0.05$, **e** D20C[20-80]T2.5N10S10I1.25 min_supp $= 0.1$, **f** D10C[20-80]T20N5S6I4 min_supp $= 0.4$

7.3 Results: efficiency of CloFAST on real datasets

The results obtained on real datasets generally confirm the observations drawn from the experiments performed on synthetic datasets. In particular, the running times shown in Fig. 13 confirm that CloFAST outperforms all the other methods when the support threshold is low, i.e., the number of frequent patterns is high. In particular, for MSNBC, MSNBC* and Snake*, which are the densest datasets (see Table 3), CloFAST clearly shows the best performance in running time. We note that for the datasets Pumbs and Pumbs*, it is difficult to appreciate the difference between CloFAST, ClaSP and CloSpan, since the high running time of BIDE

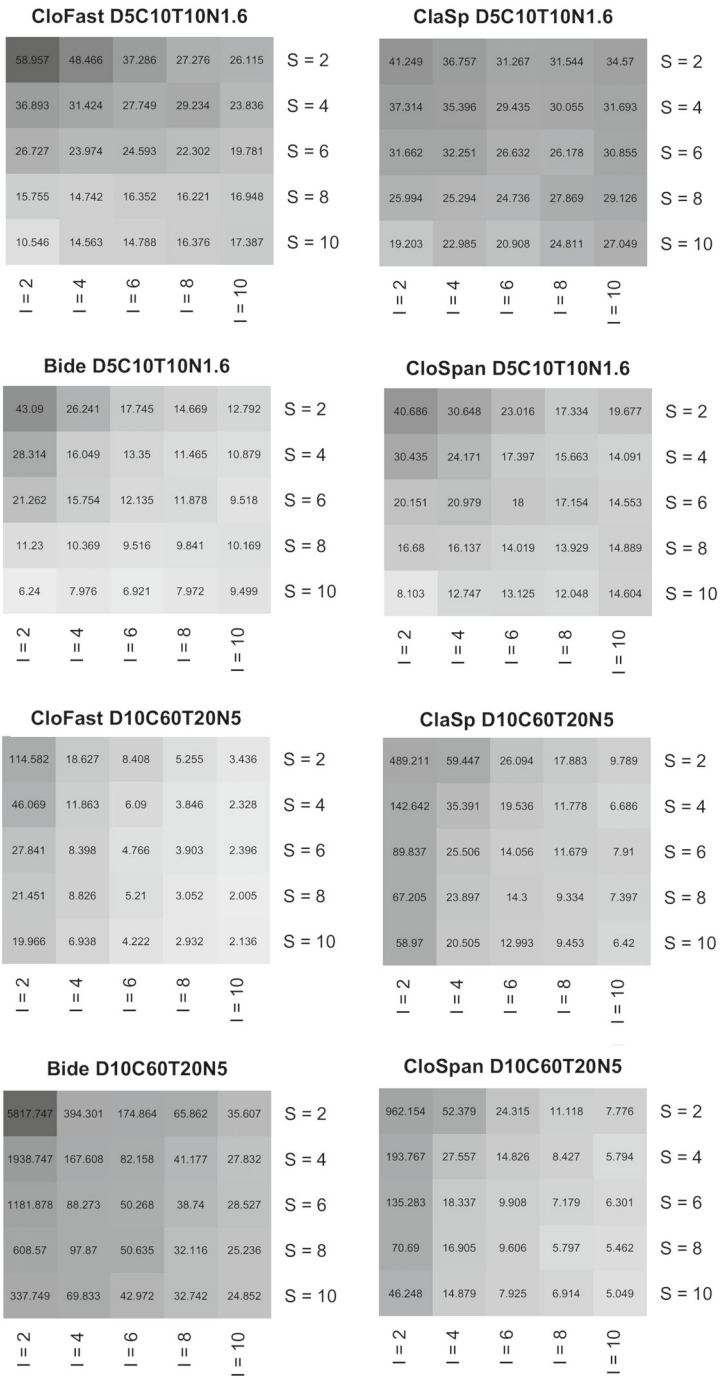


Fig. 11 Running times (in seconds) varying $S = \{2, 4, 6, 8, 10\}$ and $I = \{2, 4, 6, 8, 10\}$. Results are obtained with $D = 5$, $C = 10$, $T = 10$, $N = 1.6$, min_sup $= 0.05$ (small and sparse); $D = 10$, $C = 60$, $T = 20$, $N = 5$, min_sup $= 0.7$ (dense)

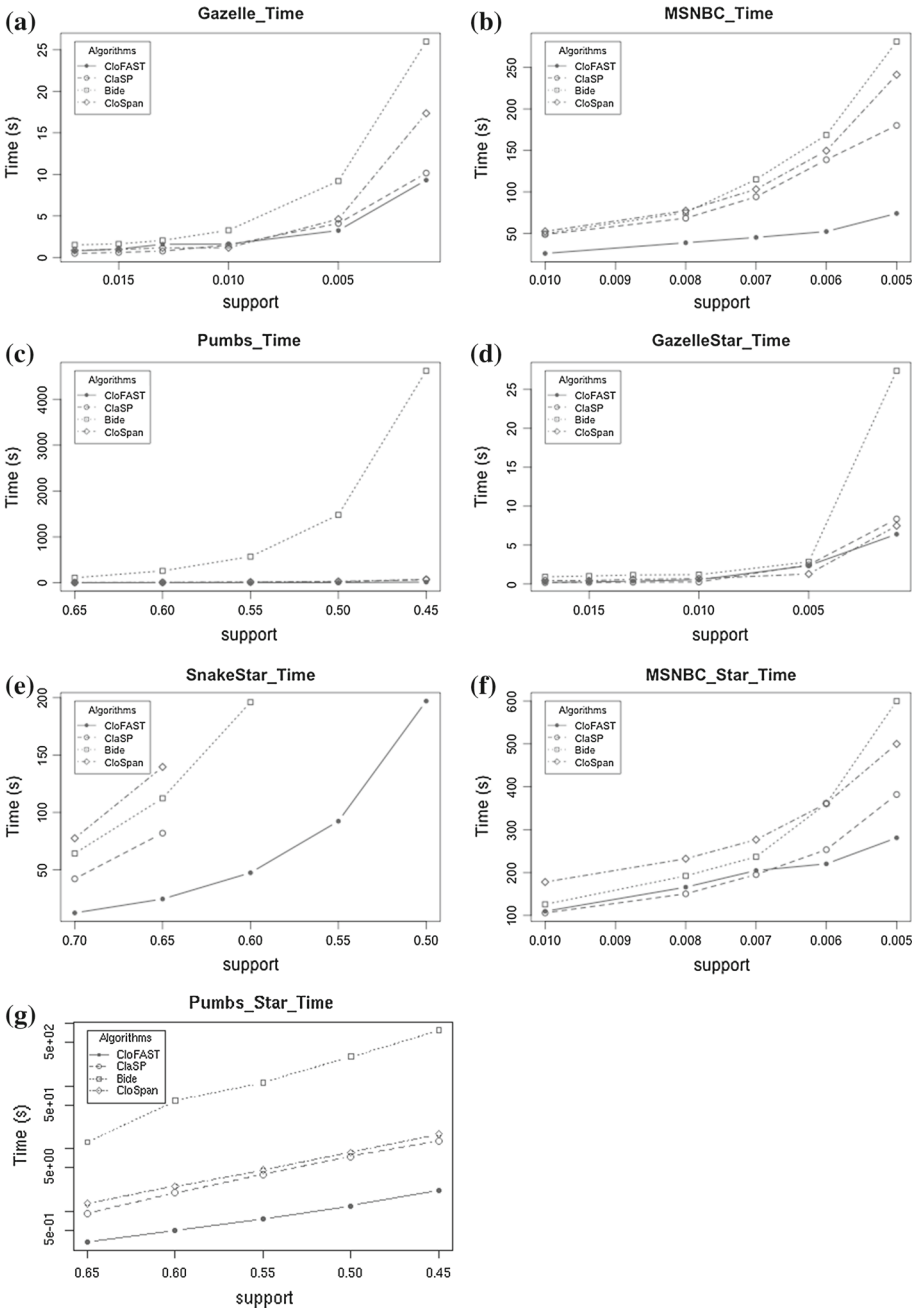


Fig. 12 Real datasets: running times. Missing values correspond to out-of-memory errors (>32 GB). **a** Gazelle, **b** MSNBC, **c** Pumsb, **d** Gazelle*, **e** Snake*, **f** MSNBC*, **g** Pumbs* (logarithmic scale)

flattes the other results. Nevertheless, we confirm that for these two datasets, CloFAST is the fastest algorithm (Fig. 12).

Concerning memory consumption, CloFAST is among the best performing methods for almost all the datasets (see Fig. 14). Some differences between CloFAST and ClaSP can be appreciated for two datasets with a large number of closed patterns (294,386 for MSNBC* with min_sup $= 0.005$, 1,300,529 for Snake* with min_sup $= 0.5$). Both algorithms use a vertical representation of the data. However, a closer look at the results reveals that, while for MSNBC* CloFAST is at a disadvantage compared to ClaSP, due to the large number of null values stored in VILs, for Snake* our internal representation is effective and CloFAST is the only method which does not incur in out-of-memory errors (the limit is 32 GB).

7.4 Scalability

In order to evaluate the scalability of CloFAST with respect to other competitive systems, we performed experiments on synthetic datasets by varying the number of input sequences. In particular, by keeping constant other parameters of the data generator ($C = 20$, $T = 20$ and $N = 2.5$), we varied D, obtaining datasets with a different number of sequences (i.e., the following configurations were used: D50C20T20N2.5S6I4, D100C20T20N2.5S6I4, D150C20T20N2.5S6I4, D200C20T20N2.5S6I4, D250C20T20N 2.5S6I4, D300C20T20N2.5S6I4).

The results shown in Fig. 15 indicate that by increasing the number of sequences, CloFAST significantly outperforms ClaSP and BIDE, both in terms of running time and in terms of memory consumption. The comparison between CloFAST and CloSpan reveals that CloFAST outperforms CloSpan (although the difference is not impressive) in terms of running time. As concerns memory consumption, CloFAST outperforms CloSpan only when the number of sequences is less than 150,000. This is not surprising, since the value of the density is not high ($T/N = 8$), and CloFAST is more effective when the density increases (see Fig. 9).

7.5 Effectiveness of closure checking and pruning

In this subsection, we investigate the effectiveness of the closure checking and of the pruning strategy implemented in CloFAST and when they come into play. To this aim, we consider three real-world datasets (i.e., Snake*, Pumbs and Pumbs*) and four synthetic datasets with different characteristics . We compare the results of CloFAST with those of FAST [19], which does not perform closure checking and pruning (Fig. 16).

The results reported in Fig. 16 confirm, as expected, that CloFAST is able to prune a higher percentage of frequent sequences when input sequences are long and when the number of input sequences increases (Snake* vs. Pumbs/Pumbs*). By comparing the results obtained with Pumbs and Pumbs*, we notice that benefits of CloFAST are more evident when the number of input sequences increases (the main difference between Pumbs and Pumbs* is in the number of sequences). Obviously, the percentage of pruned frequent sequences directly reflects on the running time and on the memory consumption. In particular, when the difference between the number of closed patterns and the number of frequent patterns increases, CloFAST shows a proportional improvement in terms of both running times and memory consumption. This is more evident for small values of the support threshold.

Experiments on synthetic datasets aimed at evaluating how the closure check and pruning performs when the number of frequent sequences in the underlying model changes. In particular, by keeping unchanged the values of D, C, T and N, we can compare the results obtained with different values of S and I, which regulate the average length of maximal sequences

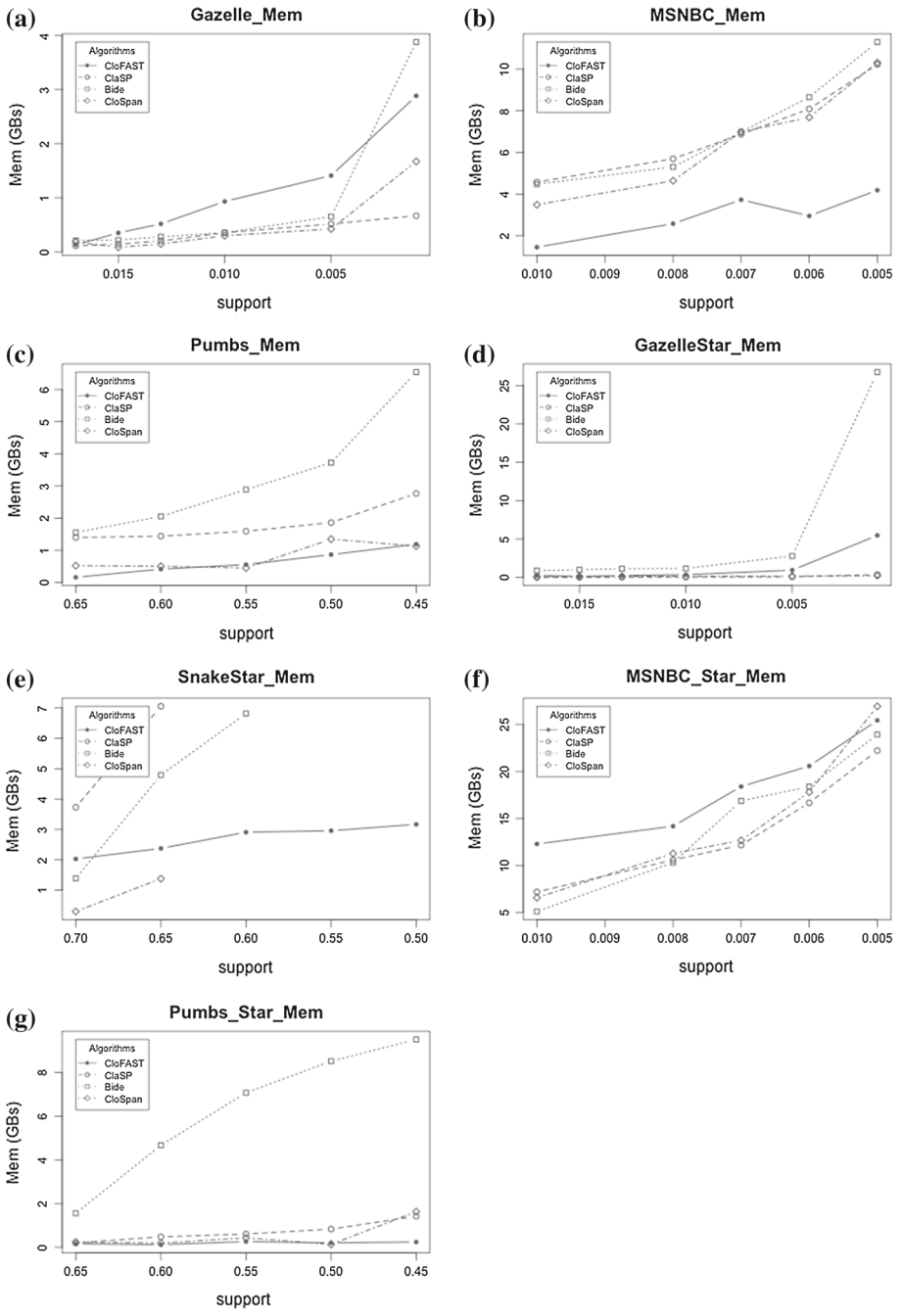


Fig. 13 Real datasets: memory consumption. Missing values correspond to out-of-memory errors (>32 GB). **a** Gazelle, **b** MSNBC, **c** Pumsb, **d** Gazelle*, **e** Snake*, **f** MSNBC*, **g** Pumbs*

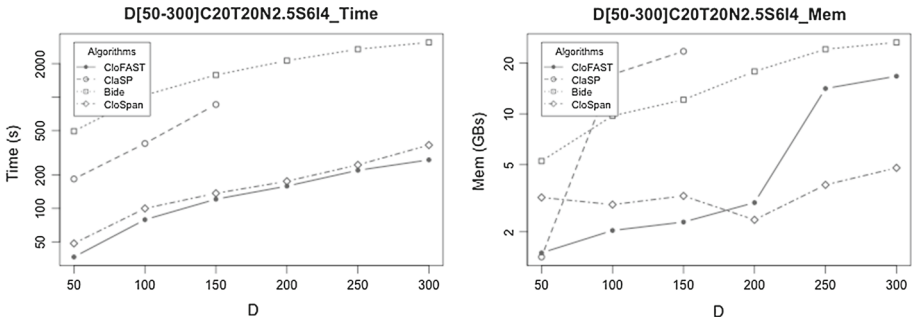


Fig. 14 Running times (in seconds) and memory consumption (in GB) varying $D = 50, 100, 150, 200, 250, 300$. Results are obtained with $C = 20$, $T = 20$, $N = 2.5$ and min_sup $= 0.4$. Missing values correspond to out-of-memory errors (>32 GB)

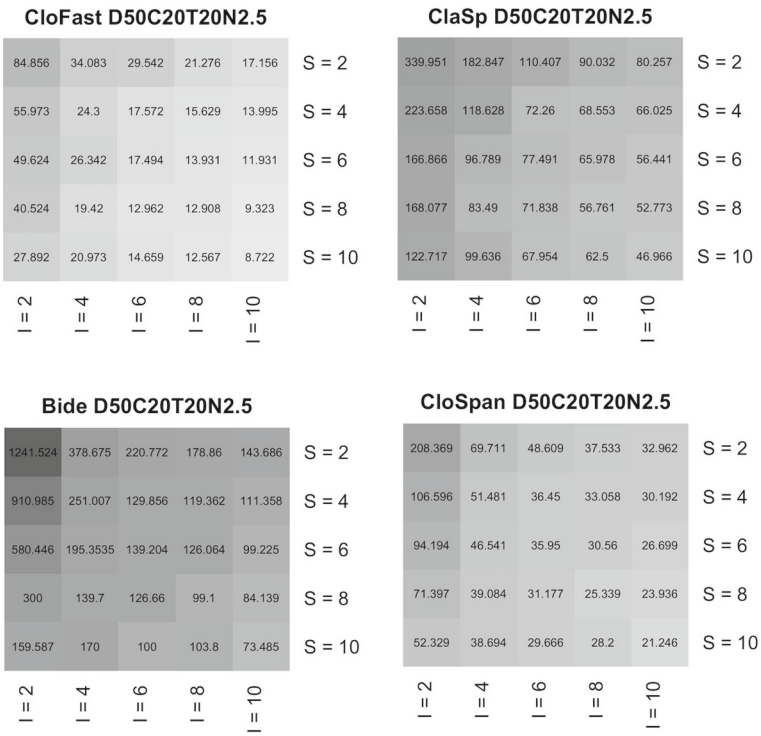


Fig. 15 Running times (in seconds) varying $S = \{2, 4, 6, 8, 10\}$ and $I = \{2, 4, 6, 8, 10\}$. Results are obtained with $D = 50$, $C = 20$, $T = 20$, $N = 2.5$, min_sup $= 0.4$

and the average size of itemsets in maximal sequences, respectively. When S is small and I is large, the underlying patterns are very similar to each other and multiple sequences covered by the same pattern are likely to be generated, thus leading to better opportunities for pruning. The results (see Fig. 17) show that with a small value of S and high value of I, CloFAST is able to prune the search space significantly, thus greatly outperforming FAST in terms of running times, at minor additional memory costs.

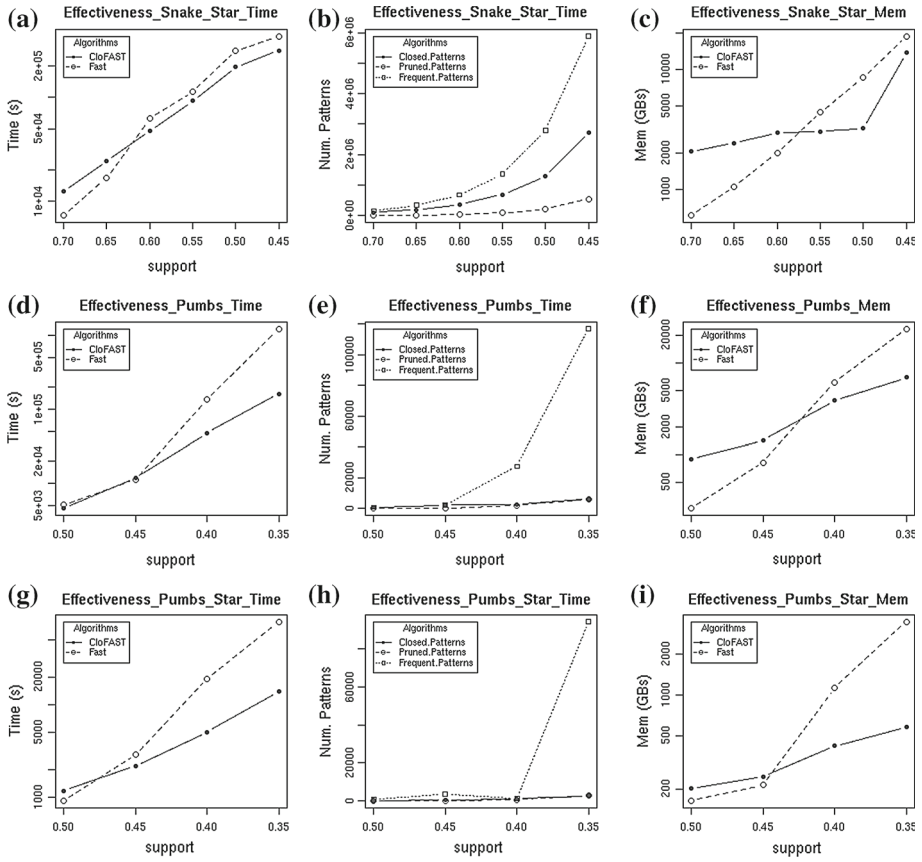


Fig. 16 Fast and CloFast comparison on real datasets. **a** Snake* time, **b** Snake* patterns, **c** Snake* memory, **d** Pumbs time, **e** Pumbs patterns, **f** Pumbs memory, **g** Pumbs* time, **h** Pumbs* patterns, **i** Pumbs* memory

8 Conclusions

In this paper, we have presented CloFAST, a novel algorithm for mining closed frequent sequences without candidate maintenance. It exploits (i) *sparse id-lists* and *vertical id-lists* for fast counting the support of sequential patterns and (ii) a novel one-step technique to check *sequence closure* and to *prune* the search space. In this way, CloFAST is also able to mine long closed sequences by reducing the effort required for space exploration, support counting and search space pruning.

A thorough experimental study with both artificial and real datasets shows that CloFAST outperforms in running times the state-of-the-art algorithms, especially for low support values and for dense datasets, i.e., when the number of frequent sequences is high. Notably, the time efficiency of CloFAST is not obtained at the cost of higher memory consumption, which remains comparable to, if not better than, that of other systems (e.g., CloSPAN). For some critical datasets, CloFAST is the only system that returns a result within the fixed memory size constraints. A comparison with its predecessor FAST, which does not perform closure checking and pruning, shows that CloFAST is more efficient both in time and memory

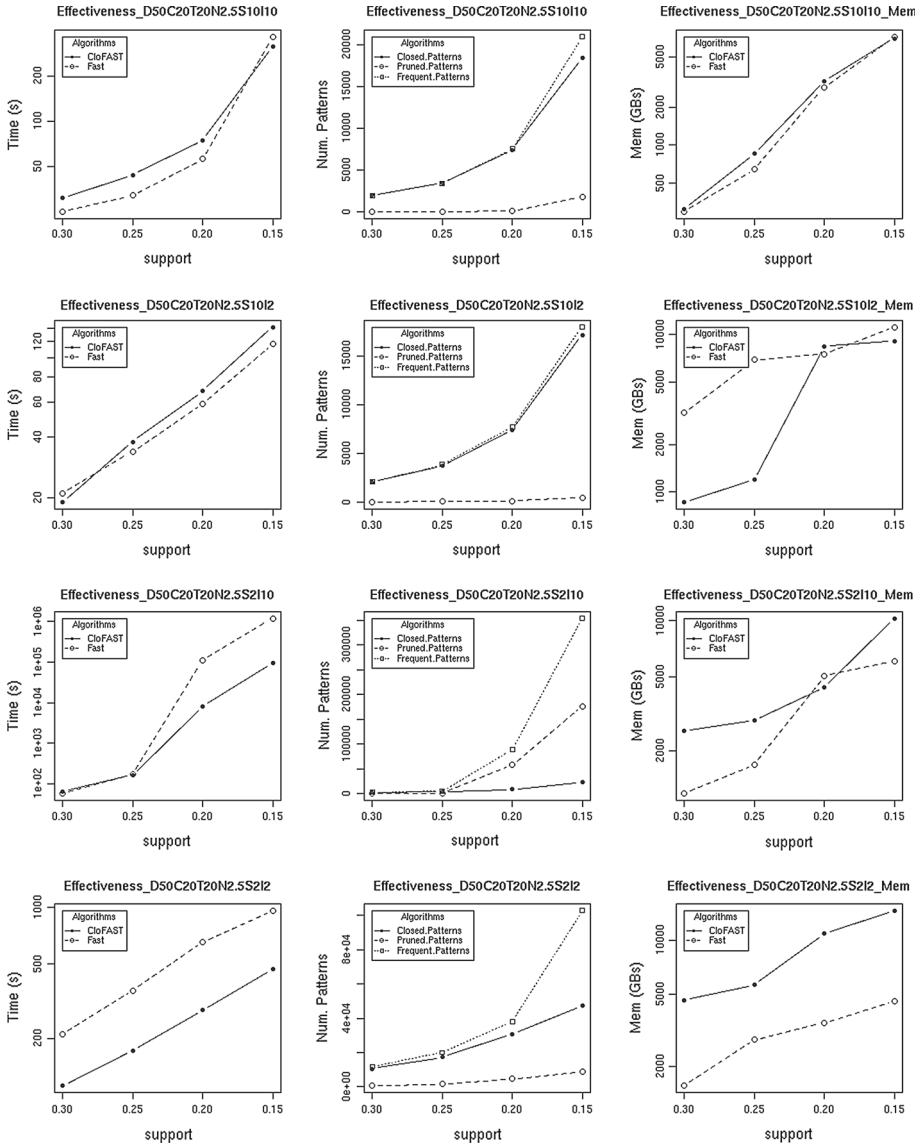


Fig. 17 Fast and CloFast comparison on synthetic datasets

consumption, thus providing a way to compact results, while preserving the same expressive power of discovered patterns.

Our work opens up some important avenues for future work. In particular, CloFAST can be profitably used in sequence prediction, by exploiting descriptive patterns for prediction purposes, as in associative classification [4]. This extension would provide an alternative way to face sequence classification both in biological domains [6] and in process mining applications [5], which are characterized by either very long sequences dense datasets. An

additional extension of CloFAST could be in the explicit consideration of noise in data, similarly to what was suggested in [7].

Acknowledgments We thank P. Fournier-Viger for kindly providing the Snake dataset. We also thank Lynn Rudd for reading through the paper. We would like to acknowledge the support of the European Commission through the project MAESTRA—Learning from Massive, Incompletely Annotated, and Structured Data (Grant number ICT-2013-612944). Finally, this work is in partial fulfillment of the requirements of the Italian project VINCENTE PON02 00563 3470993 "A Virtual collective INtelligenCe ENvironment to develop sustainable Technology Entrepreneurship ecosystems." The authors also wish to thank Lynn Rudd for her help in reading the manuscript.

References

1. Agrawal R, Srikant R (1995) Mining sequential patterns. In: Proceedings of the eleventh international conference on data engineering, ICDE '95. IEEE Computer Society, Washington, DC, pp 3–14
2. Ayres J, Flannick J, Gehrke J, Yiu T (2002) Sequential pattern mining using a bitmap representation. In: Proceedings of the eighth ACM SIGKDD international conference on knowledge discovery and data mining, KDD '02. ACM, New York, NY, pp 429–435
3. Burdick D, Calimlim M, Flannick J, Gehrke J, Yiu T (2005) MAFIA: a maximal frequent itemset algorithm. IEEE Trans Knowl Data Eng 17(11):1490–1504
4. Ceci M, Appice A (2006) Spatial associative classification: propositional vs structural approach. J Intell Inf Syst 27(3):191–213
5. Ceci M, Lanotte PF, Fumarola F, Cavallo DP, Malerba D (2014) Completion time and next activity prediction of processes using sequential pattern mining. In: Dzeroski S, Panov P, Kocev D, Todorovski L (eds) Discovery science—17th international conference, DS 2014, Bled, Slovenia, October 8–10, 2014. Proceedings, volume 8777 of Lecture Notes in Computer Science, Springer, pp 49–61
6. Ceci M, Loglisci C, Salvemini E, D'Elia D, Malerba D (2011) Mining spatial association rules for composite motif discovery. In: Bruni R (ed) Mathematical approaches to polymer sequence analysis and related problems. Springer, Berlin, pp 87–109
7. Cerf L, Besson J, Nguyen K-N, Boulicaut J-F (2013) Closed and noise-tolerant patterns in n-ary relations. Data Min Knowl Discov 26(3):574–619
8. Chi Y, Wang H, Yu PS, Muntz RR (2006) Catch the moment: maintaining closed frequent itemsets over a data stream sliding window. Knowl Inf Syst 10:265–294
9. Exarchos TP, Tsipouras MG, Papaloukas C, Fotiadis DI (2008) A two-stage methodology for sequence classification based on sequential pattern mining and optimization. Data Knowl Eng 66:467–487
10. Fournier-Viger P (2014) SPMF: a sequential pattern mining framework. http://www.philippe-fournier-viger.com/spmf/index.php. Accessed 08 Aug 2014
11. Fradkin D, Moerchen F (2010) Margin-closed frequent sequential pattern mining. In: Proceedings of the ACM SIGKDD workshop on useful patterns, UP '10. ACM, New York, NY, pp 45–54
12. Gomariz A, Campos M, Marín R, Goethals B (2013) ClaSP: an efficient algorithm for mining frequent closed sequences. In: Pei J, Tseng VS, Cao L, Motoda H, Xu G (eds) PAKDD (1), vol 7818 of Lecture Notes in Computer Science. Springer, Berlin, pp 50–61
13. Han J (2005) Data mining: concepts and techniques. Morgan Kaufmann Publishers Inc., San Francisco
14. Huang K-Y, Chang C-H, Tung J-H, Ho C-T (2006) COBRA: closed sequential pattern mining using bi-phase reduction approach. In: Tjoa AM, Trujillo J (eds) DaWaK, vol 4081 of Lecture Notes in Computer Science. Springer, Berlin, pp 280–291
15. Jingjun Zhu GG, Wu Haiyan (2010) An efficient method of web sequential pattern mining based on session filter and transaction identification. J Netw 5(9):1017–1024
16. Li Z, Lu S, Myagmar S, Zhou Y (2006) Cp-miner: finding copy-paste and related bugs in large-scale software code. IEEE Trans Softw Eng 32:176–192
17. Masseglia F, Poncelet P, Teisseire M (2009) Efficient mining of sequential patterns with time constraints: reducing the combinations. Expert Syst Appl Int J 36:2677–2690
18. Pei J, Han J, Mortazavi-Asl B, Pinto H, Chen Q, Dayal U, Hsu M (2001) PrefixSpan: mining sequential patterns by prefix-projected growth. In: Proceedings of the 17th international conference on data engineering. IEEE Computer Society, Washington, DC, pp 215–224
19. Salvemini E, Fumarola F, Malerba D, Han J (2011) FAST sequence mining based on sparse id-lists. In: Kryszkiewicz M, Rybinski H, Skowron A, Ras ZW (eds) ISMIS, vol 6804 of Lecture Notes in Computer Science, Springer, Berlin, pp 316–325

20. Song S, Hu H, Jin S (2005) HVSM: a new sequential pattern mining algorithm using bitmap representation. In: Li X, Wang S, Dong Z (eds) Advanced Data Mining and Applications, vol 3584, Lecture Notes in Computer ScienceSpringer, Berlin Heidelberg, pp 455–463
21. Turi A, Loglisci C, Salvemini E, Grillo G, Malerba D, D'Elia D (2009) Computational annotation of UTR cis-regulatory modules through frequent pattern mining. BMC Bioinform 10:1–12. doi:10.1186/1471-2105-10-S6-S25
22. Wang J, Han J, Li C (2007) Frequent closed sequence mining without candidate maintenance. IEEE Trans. Knowl. Data Eng. 19:1042–1056
23. Yan X, Han J, Afshar R (2003) CloSpan: mining closed sequential patterns in large datasets. In: SDM, pp 166–177
24. Yang Z, Kitsuregawa M (2005) LAPIN-SPAM: an improved algorithm for mining sequential pattern. In: 22nd international conference on data engineering workshops, vol 0, pp 1222
25. Zaki MJ (2001) SPADE: an efficient algorithm for mining frequent sequences. Mach Learn 42(1–2):31–60
26. Zhang X, Dong G, Ramamohanarao K (2000) Exploring constraints to efficiently mine emerging patterns from large high-dimensional datasets. In: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '00). ACM, New York, 310–314. http://dx.doi.org/10.1145/347090.347158

Fabio Fumarola Ph.D., is a research assistant at the Department of Computer Science of the University of Bari, Italy. He was visiting researcher at the University of Illinois at Urbana Champaign. He also worked as lead data scientist at Angelo Investment Group on big data applied to game analytics. He was awarded a prize by the Apulia Region for the project "WhereToLive" on social security monitoring through big data mining solutions. His research mainly concerns data stream mining, web information harvesting, and big data. He co-authored more than 20 papers on referred journal and conferences and a book on Data Mining Techniques in Sensor Networks.

Pasqua Fabiana Lanotte is a Ph.D. student of Computer Science at University of Bari, Italy. Her research interests are sequential pattern mining, Web mining and information extraction. She received her Master degree in Computer Science at University of Bari, and she is currently a visiting scholar of Computer Science at University of Illinois, Urbana-Champaign. She was awarded a prize by the Apulia Region for the project "WhereToLive" on social security monitoring through big data mining solutions.

Michelangelo Ceci Ph.D., is an assistant professor at the Department of Computer Science, University of Bari, Italy. His main research interests are in data mining and machine learning. He was a visiting researcher at the University of Bristol (U.K.) and at the JSI (SLO). He has published more than 140 papers in refereed journals and conferences. He is responsible for a research unit of the MAESTRA EU project and of several national projects. He has served in the Program Committee of many conferences, including: IEEE ICDM, IJCAI, ECMLPKDD. He is member of the editorial boards of: IJSNM, IJDSN, IJDATS and JAIS. He has been the program co-chair of five workshops and DS2016, the organizing committee chair of SEBD 2007 and member of the editorial board of the ECMLPKDD 2014 and 2015 journal tracks.

Donato Malerba is a full professor at the Department of Computer Science of the University of Bari Aldo Moro. His research activity mainly concerns data mining, machine learning, data science and big data. He has published more than 200 papers in international journals and conference proceedings. He received the IBM Faculty Award for the year 2004. He is responsible for a research unit of several European and national projects. He is the Director of the CINI National Lab on Big Data and a member of the Board of Directors of the Big Data Value Association.