Machine Learning (2020) 109:1147–1177
https://doi.org/10.1007/s10994-019-05827-w

Effective approximation of parametrized closure systems over transactional data streams

Daniel Trabold[1,3] · **Tamás Horváth**[1,2,3] · **Stefan Wrobel**[1,2,3]

Received: 17 June 2018 / Revised: 9 February 2019 / Accepted: 6 July 2019 / Published online: 30 August 2019

Abstract
Strongly closed itemsets, defined by a parameterized closure operator, are a generalization of ordinary closed itemsets. Depending on the strength of closedness, the family of strongly closed itemsets typically forms a tiny subfamily of ordinary closed itemsets that is stable against changes in the input. In this paper we consider the problem of mining strongly closed itemsets from transactional data streams. Utilizing their algebraic and algorithmic properties, we propose an algorithm based on reservoir sampling for approximating this type of itemsets in the landmark streaming setting, prove its correctness, and show empirically that it yields a considerable speed-up over a straightforward naive algorithm without any significant loss in precision and recall. We motivate the problem setting considered by two practical applications. In particular, we first experimentally demonstrate that the above properties, i.e., compactness and stability, make strongly closed itemsets an excellent indicator of certain types of concept drifts in transactional data streams. As a second application we consider computer-aided product configuration, a real-world problem raised by an industrial project. For this problem, which is essentially exact concept identification, we propose a learning algorithm based on a certain type of subset queries formed by strongly closed itemsets and show on real-world datasets that it requires significantly less query evaluations than a naive algorithm based on membership queries.

Keywords Data streams · Itemset mining · Closed itemsets · Concept drift detection · Applications

✉ Daniel Trabold
Daniel.Trabold@iais.fraunhofer.de

Tamás Horváth
Tamas.Horvath@iais.fraunhofer.de

Stefan Wrobel
Stefan.Wrobel@iais.fraunhofer.de

[1] Fraunhofer IAIS, Schloss Birlinghoven, 53754 Sankt Augustin, Germany

[2] Department of Computer Science, University of Bonn, Bonn, Germany

[3] Fraunhofer Center for Machine Learning, Sankt Augustin, Germany

1 Introduction

It is a well-known fact that *closed* frequent itemsets provide a compact representation of frequent itemsets (Boros et al. 2003; Pasquier et al. 1999). The concept of closedness has been generalized in Boley et al. (2009): An itemset is *strongly* or more precisely, Δ-*closed* for some integer $\Delta > 0$, if all of its extensions result in a drop of at least Δ transactions in its support set. Clearly, Δ-closed itemsets are ordinary closed (i.e., 1-closed) for any $\Delta \geq 1$. With increasing Δ, the number of Δ-closed itemsets becomes usually much smaller than that of ordinary closed itemsets (Boley et al. 2009) (i.e., the number of patterns can be controlled by Δ). The most common way to control the size of the output pattern set is to eliminate patterns by some frequency threshold. This strategy shifts, however, the language bias towards short patterns. Assuming a *fixed* capacity of maintaining at most K patterns, the length of a longest frequent itemset is bounded by $\log_2 K$. In contrast, Δ-closed itemsets may include long patterns as well. We also note that the set of strongly (and hence, ordinary) closed itemsets can further be reduced by pruning with frequency. We omit this straightforward generalization for simplicity.

Despite the fact that strongly closed itemsets provide only lossy representations of frequent itemsets, this typically *tiny* subset of ordinary closed itemsets is still able to capture some *essential* information about the data at hand that is *stable* against changes (Boley et al. 2009). Compactness and stability make strongly closed itemsets attractive, among others, for streaming applications, motivating us to consider the problem of mining strongly closed itemsets in transactional data streams. In addition to these advantageous properties, strongly closed itemsets can be generated efficiently (i.e., with polynomial delay, Ganter and Reuter 1991; Gély 2005), in contrast to e.g. maximal frequent itemsets (which, unless P = NP, cannot be listed in output polynomial time, Boros et al. 2003).

The properties mentioned above constitute a further strength of strongly closed itemsets for such data stream scenarios where there is some fixed upper bound K on the number of output patterns. Such a bound can be derived e.g. from time or space constraints and allows an *automatic* adjustment of the parameter Δ as a function of K during the enumeration. For example, our algorithm can be modified in such a way that once the number of patterns exceeds K, it immediately increases the value of Δ by the *smallest* amount such that the number of "surviving" patterns will be at most K.

As a first practical application of our approach, we consider the *transactional* data stream scenario in which the objects arriving continuously are subsets of some ground set (set of all items) and generated by some *unknown* distribution that may change over time. Such changes are referred to as *concept drifts*. Concept drift detection is essential for most algorithms building some model from data streams with changing distributions. With an extensive series of experiments we demonstrate that changes in the family of strongly closed itemsets are excellent indicators for detecting concept drifts in transactional data streams. As an example, we could reliably detect concept drifts by monitoring the changes in around 250 strongly closed itemsets out of more than 45,000 ordinary closed ones.

As a second application, we consider the problem of computer-aided *product configuration*. For this problem we assume that there is some dynamic transactional database (i.e., a transactional data stream) over a finite set of items, where the transactions correspond to individually customized products purchased by customers. The goal is then to support the next customer in selecting the subset of items constituting her desired product to be purchased, without showing her all possible items one by one, i.e., without asking membership queries in a brute-force manner. We propose an algorithm for this problem that, in contrast to the brute-force strategy, asks a certain type of *subset* queries. These subset queries are based on

strongly closed itemsets extracted from the data stream of transactions already seen. Using a database of product configuration transactions from a real-world industrial project, we show that even 50% of the membership queries asked by the brute-force algorithm can be saved for the average user by a simple algorithm[1] based on strongly closed itemsets.

Motivated by these and other practical applications, we present an efficient algorithm for mining strongly closed itemsets from transactional data streams in the *landmark* model. To make the algorithm practically feasible for *massive* transactional data streams, we consider a random subset of the data stream generated by *reservoir sampling* (Vitter 1985) and approximate the family of strongly closed itemsets in the data stream by that in the sample. The size of the sample is chosen in a way that with high probability, it preserves the relative frequencies of itemsets within some small error. Our algorithm calculates the family of strongly closed itemsets from this sample upon request or after a certain number of new transactions have arrived since the last update.

Reservoir sampling allows us to record the changes from the last update not in the sample, but in two separate databases. As the replacement of a transaction in the sample is equivalent to removing the old transaction and inserting the new one, the two databases correspond to the sets of transactions to be *deleted* from and those to be *added* to the sample. The motivation behind splitting replacement into deletion and insertion is that in contrast to the method in Boley et al. (2009), strong closedness of an itemset can be decided much faster when the support set of the itemset is empty in at least one of the two databases. With increasing stream length this situation becomes more and more typical as the number of changes in the sample and accordingly, the size of the two databases decreases.

Our algorithm is based on the fact that strongly closed itemsets of the sample form a *closure system* (Boley et al. 2009). We make use of this property and calculate the update by traversing the old strongly closed itemsets with a divide and conquer algorithm. It is based on a folklore algorithm (see, e.g., Gély 2005) that lists all closed sets of a set system. This algorithm has a number of advantageous algorithmic properties (Boley et al. 2010; Gély 2005) utilized by our algorithm as well.

We empirically evaluated the *speed-up* and *quality* (in terms of precision and recall) of our algorithm on artificial and real-world benchmark datasets. To measure the speed-up, we compared the batch algorithm generating strongly closed itemsets from the new sample from scratch with our incremental algorithm for different number of changes in the sample. For small changes, which is the case for long data streams, we obtained a speed-up of up to two orders of magnitude. Regarding the quality, in most cases we achieved very high precision and recall values (close to 1). Thus, as the empirical results demonstrate, our algorithm is much faster than the algorithm computing from scratch and still calculates a close approximation of the family of strongly closed itemsets.

An early version of this paper appeared in Trabold and Horváth (2017). The main differences between the two versions are in the empirical evaluation of the algorithm. In particular, for concept drift detection in this extended version we additionally analyze the effects of different concept drift types, drift lengths, varying similarity for intersected drifts, various delays between instances of our algorithm, and the buffer size of our algorithm. In addition to concept drift detection, in this version we also consider the application of strongly closed

[1] Our main focus in this work was on mining strongly closed itemsets from transactional data streams and on demonstrating the potential of this kind of itemsets on real-world data streaming problems. Accordingly, a more sophisticated optimized version of our algorithms utilizing some domain specific properties goes far beyond the scope of this paper.

itemsets to transaction identification with subset queries. For this problem we present an algorithm and evaluate it empirically on various real-world product configuration datasets.

Outline The rest of the paper is organized as follows. We briefly discuss related work in Sect. 2, define the necessary concepts and the problem setting in Sect. 3, and describe our algorithm in Sect. 4. In Sect. 5 we empirically evaluate the speed-up and approximation quality of our algorithm on various benchmark and real-world datasets. In Sect. 6 we present extensive experimental results demonstrating the suitability of strongly closed itemsets for concept drift detection and for computer-aided product configuration. We finally mention some interesting directions for future research in Sect. 7.

2 Related work

Mining ordinary closed itemsets is the special case of mining Δ-closed itemsets for $\Delta = 1$. Out of the different streaming models (e.g., sliding window, time-fading, landmark) studied for mining closed frequent itemsets in data streams, we discuss only algorithms for the *landmark* model, corresponding to our problem setting. For a survey on closed frequent itemset mining in data streams, the reader is referred to Bai and Kumar (2016).

The vast majority of literature on this subject considers the *sliding window* model, in which a window of fixed size is moved forward as new transactions arrive in a data stream. The window always contains the most recent k transactions, where k is the window size defined by the user. The most prominent algorithms for this setting are Moment (Chi et al. 2004) and CFI-Stream (Jiang and Gruenwald 2006). In the *time-fading model* all transactions have an associated weight, which is high for recent transactions and decays over time such that transactions seen long ago will be eventually removed from consideration. Algorithms for this model include WSW (Tsai 2009) and CLICI (Gupta et al. 2010).

The *landmark model* considers all transactions from a landmark starting time in the past to the current time point. The particular challenge for closed itemset mining in this setting is that more and more sets will become closed as the data stream gets larger. For very long streams every set will become closed eventually with high probability, resulting in a huge output. The algorithms FP-CDS (Liu et al. 2009) and LC-CloStream (Iwanuma et al. 2016) mine ordinary closed frequent itemsets under the landmark model. To the best of our knowledge these are the only algorithms concerning this problem setting. As strongly closed itemsets generalize ordinary ones, we consider a more general problem that has not been studied before.

FP-CDS (Liu et al. 2009) processes the transactions in batches. It first constructs a local tree structure for the current batch of transactions and then merges this tree with a global one built for the entire data stream from the landmark starting time up to the previous batch. Closed patterns are generated from this global tree from scratch. Using the idea in Manku and Motwani (2002), the algorithm considers all patterns of frequency at least ϵ for some appropriately chosen $\epsilon < \theta$, where θ is the frequency threshold. Our approach is fundamentally different, as we process the transactions with reservoir sampling. Furthermore, while our algorithm incrementally updates the family of closed itemsets, FP-CDS computes it from scratch.

LC-CloStream (Iwanuma et al. 2016) combines the main features of two data stream algorithms, LossyCounting (Manku and Motwani 2002) mining frequent itemsets in the landmark model and CloStream (Yen et al. 2011) mining closed frequent itemsets in the sliding window model. Similarly to LossyCounting, LC-CloStream computes an ϵ-approximation of

ordinary closed frequent itemsets and similarly to CloStream, it calculates ordinary closed itemsets from intersections of transactions. LC-CloStream returns all closed itemsets which are estimated to be frequent. Its output is incomplete, in contrast to ours, which is complete with respect to the transactions in the reservoir.

Finally we note that there is a vast amount of literature on concept drift detection in data streams (see, e.g., Gama et al. 2014 for a survey). Several authors (see, e.g., Kifer et al. 2004; van Leeuwen and Siebes 2008) specifically consider the problem of concept drift detection in transactional data streams. Since concept drift detection is only one of the potential applications of our *general-purpose* algorithm, we omit the discussion of the related literature and mention only the KARMA algorithm (Loglisci et al. 2018) designed for concept drift detection in *network* data streams. Similarly to our approach, it is based on pattern mining from transactional data streams.[2] However, while KARMA defines interestingness by frequency, we consider an entirely different set of patterns specified by Δ-closedness. Whereas the size of the output pattern set is difficult to control by frequency during the mining process, we can govern it effectively by the appropriate choice of Δ. We also note that KARMA can be extended to patterns defined by Δ-closed itemsets.

3 The problem setting

In this section we define the problem setting for this work. We first provide the necessary notions and fix the notation. For all $m \in \mathbb{N}$, $[m]$ denotes the set $\{1, \ldots, m\}$. Given some finite ground set E (*items*), the concepts of *itemsets* and *transactions* (i.e., subsets of E), and *transaction databases* over E (i.e., multisets of transactions) are used in the standard way. A *transactional data stream* over E (in what follows, simply a *data stream*) is a sequence $\mathcal{S}_t = \langle T_1, T_2, \ldots, T_t \rangle$, where the T_i's are non-empty transactions, i.e., $\emptyset \neq T_i \subseteq E$ for all $i \in [t]$. To calculate the family of strongly closed itemsets for $\mathcal{S}_t$, the order of the transactions in $\mathcal{S}_t$ does not matter. Therefore, $\mathcal{S}_t$ can be regarded for this operation as a transaction database (i.e., multi-set) $\mathcal{D}_t$ over E. We make use of this property and formulate most definitions below for transaction databases.

Let $\mathcal{D}$ be a transaction database over E. The *support set* of an itemset $X \subseteq E$ in $\mathcal{D}$, denoted $\mathcal{D}[X]$, is defined by the multi-set $\{T \in \mathcal{D} : X \subseteq T\}$; the *support count* of X by the cardinality $|\mathcal{D}[X]|$ of $\mathcal{D}[X]$. For a threshold $\tilde{\Delta} \in [0, 1]$, an itemset $X \subseteq E$ is *relatively* $\tilde{\Delta}$*-closed* in $\mathcal{D}$ if

$$\frac{|\mathcal{D}[X]|}{|\mathcal{D}|} - \frac{|\mathcal{D}[Y]|}{|\mathcal{D}|} \geq \tilde{\Delta} \tag{1}$$

holds for all Y with $X \subsetneq Y \subseteq E$. That is, any proper extension of X decreases its relative frequency by at least $\tilde{\Delta}$. Thus, $\tilde{\Delta}$ indicates the *strength* of the closure. If it is clear from the context, we omit the adverb "relatively". Motivated by different real-world applications (e.g., concept drift detection, computer aided product configuration), we consider the following mining problem:

> **$\tilde{\Delta}$-Closed Set Listing Problem** *Given* a single pass over a transactional data stream $\mathcal{S}_t = \langle T_1, T_2, \ldots, T_t \rangle$ over a set E of items (i.e., $T_i \subseteq E$ for all $i \in [t]$), a threshold $\tilde{\Delta} \in [0, 1]$, and an integer $t' \in [t]$, *list* all itemsets $X \subseteq E$ that are $\tilde{\Delta}$-closed in $\mathcal{S}_{t'} = \langle T_1, T_2, \ldots, T_{t'} \rangle$.

[2] One of the main ideas of KARMA is the reduction of network data streams to transactional data streams by defining items with certain triples extracted from the network.

Note that the definition of *relative* $\tilde{\Delta}$-closedness for $\mathcal{S}_t$ above can equivalently be reformulated by that of *absolute* Δ-closedness (Boley et al. 2009) as follows: X is relatively $\tilde{\Delta}$-*closed* in $\mathcal{S}_t$ if and only if it is absolutely Δ-*closed* in $\mathcal{S}_t$ for $\Delta = \lceil t\tilde{\Delta} \rceil$, that is, $|\mathcal{S}_t[X]| - |\mathcal{S}_t[Y]| \geq \Delta$ for all Y with $X \subsetneq Y \subseteq E$. The adverb "absolutely" will be omitted when it is clear from the context. Clearly, ordinary closed itemsets are 1-closed itemsets. For this reason, Δ-closed itemsets will also be referred to as *strongly* closed itemsets (Boley et al. 2009) when there is no emphasis on Δ. In general, the family of Δ-closed itemsets in a transaction database $\mathcal{D}$ is denoted by $\mathcal{C}_{\Delta,\mathcal{D}}$. In particular, the family of Δ-closed itemsets in $\mathcal{S}_t$ above is denoted by $\mathcal{C}_{\Delta,\mathcal{S}_t}$. The relevance of absolute Δ-closed itemsets to our work is that we approximate the family of relative $\tilde{\Delta}$-closed itemsets in $\mathcal{S}_t$ by that in a random *sample* of $\mathcal{S}_t$ for some *fixed* size s. In this way, we can make use of some advantageous algebraic and algorithmic properties of *absolute* Δ-closed itemsets (Boley et al. 2009) and work with them for $\Delta = \lceil s\tilde{\Delta} \rceil$.

We recall some basic algebraic and algorithmic properties of Δ-closed itemsets from Boley et al. (2009). We start with the definition of closure operators. Let E be some finite set and $\sigma : 2^E \to 2^E$ be a function, where 2^E denotes the power set of E. Then σ is *extensive* if $X \subseteq \sigma(X)$, *monotone* if $X \subseteq Y$ implies $\sigma(X) \subseteq \sigma(Y)$, and *idempotent* if $\sigma(X) = \sigma(\sigma(X))$ for all $X, Y \subseteq E$. If σ is extensive and monotone then it is a *preclosure*; if, in addition, it is idempotent then it is a *closure* operator on E. A set $X \subseteq E$ is *closed* if it is a fixed point of σ (i.e., $X = \sigma(X)$).

For a transaction database $\mathcal{D}$ over E and integer $\Delta > 0$, let $\hat{\sigma}_{\Delta,\mathcal{D}} : 2^E \to 2^E$ be defined by

$$\hat{\sigma}_{\Delta,\mathcal{D}}(X) = X \cup \{e \in E \setminus X : |\mathcal{D}[X]| - |\mathcal{D}[X \cup \{e\}]| < \Delta\}$$

for all $X \subseteq E$. It holds that $\hat{\sigma}_{\Delta,\mathcal{D}}$ is a preclosure on E that is not idempotent (Boley et al. 2009). For an itemset $X \subseteq E$, consider the sequence $\hat{\sigma}^0_{\Delta,\mathcal{D}}(X), \hat{\sigma}^1_{\Delta,\mathcal{D}}(X), \hat{\sigma}^2_{\Delta,\mathcal{D}}(X), \ldots$ with $\hat{\sigma}^0_{\Delta,\mathcal{D}}(X) = X$, $\hat{\sigma}^1_{\Delta,\mathcal{D}}(X) = \hat{\sigma}_{\Delta,\mathcal{D}}(X)$, and $\hat{\sigma}^{l+1}_{\Delta,\mathcal{D}}(X) = \hat{\sigma}_{\Delta,\mathcal{D}}(\hat{\sigma}^l_{\Delta,\mathcal{D}}(X))$ for all $l \geq 1$. This sequence has a smallest fixed point, giving rise to the following definition: For all $X \subseteq E$, let $\sigma_{\Delta,\mathcal{D}} : 2^E \to 2^E$ be defined by $\sigma_{\Delta,\mathcal{D}}(X) = \hat{\sigma}^k_{\Delta,\mathcal{D}}(X)$ with

$$k = \min\{l \in \mathbb{N} : \hat{\sigma}^l_{\Delta,\mathcal{D}}(X) = \hat{\sigma}^{l+1}_{\Delta,\mathcal{D}}(X)\} \ .$$

The proof of the claims in the theorem below can be found in Boley et al. (2009).

Theorem 1 *Let $\mathcal{D}$ be a transaction database over some finite ground set E and $\Delta > 0$ an integer. Then*

(i) *for all $X \subseteq E$, X is Δ-closed in $\mathcal{D}$ if and only if $X = \sigma_{\Delta,\mathcal{D}}(X)$,*
(ii) *$\sigma_{\Delta,\mathcal{D}}$ is a closure operator over E,*
(iii) *for all $X \subseteq E$, the closure $\sigma_{\Delta,\mathcal{D}}(X)$ of X can be computed by Algorithm 1 in time $O(\|\mathcal{D}[X]\|_0)$, where $\|\mathcal{D}[X]\|_0 = \sum_{T \in \mathcal{D}[X]} |E \setminus T|$.*[3]

Using the fact that $\sigma_{\Delta,\mathcal{D}}$ is a closure operator, the family of all Δ-closed itemsets of a dataset $\mathcal{D}$ can be enumerated by the following divide and conquer folklore algorithm (see, e.g., Gély 2005): Generate first recursively all Δ-closed supersets of a set that contain a certain item $e \in E$, and then all that do not. This algorithm lists all Δ-closed itemsets non-redundantly, with polynomial delay and in polynomial space (for further properties of this algorithm see Boley et al. 2010).

[3] In Boley et al. (2009), two different algorithms are discussed for the computation of $\sigma_{\Delta,\mathcal{D}}$. They both have a time complexity of $O(\|\mathcal{D}[X]\|_0)$, assuming that empty transactions are not allowed. In practice, the two algorithms behave differently from the point of view of the running time, depending on the sparsity of $\mathcal{D}$ (cf. Boley et al. (2009) for a detailed discussion).

Algorithm 1 CLOSURE BOLEY ET AL. (2009)

input: $X \subseteq E$ and integer $\Delta > 0$
require: dataset $\mathcal{D}$ over E
output: $\sigma_{\Delta,\mathcal{D}}(X)$

1: $C \leftarrow X$; $\mathcal{D}' \leftarrow \mathcal{D}[X]$
2: **repeat**
3: **for all** $e \in E \setminus C$ **do**
4: **if** $|\mathcal{D}'| - |\mathcal{D}'[e]| < \Delta$ **then** $C \leftarrow C \cup \{e\}$; $\mathcal{D}' \leftarrow \mathcal{D}'[e]$
5: **until** $\mathcal{D}'$ has not been changed in Loop 3–4
6: **return** C

Fig. 1 Transactional data stream example

tid	items
1	bd
2	bd
3	bd
4	ad
5	ad
6	cd
7	cd
8	c
9	c

4 The mining algorithm

In this section we present our algorithm for the $\tilde{\Delta}$-Closed Set Listing problem defined in Sect. 3. To tackle massive data streams in feasible time, we approximate the $\tilde{\Delta}$-closed sets for a data stream $\mathcal{S}_t = \langle T_1, \ldots, T_t \rangle$ at time t from a random sample $\mathcal{D}_t$ generated from $\mathcal{S}_t$ without replacement. Since the order of the elements in the sample does not matter, $\mathcal{D}_t$ is regarded as a transaction database. The size s of $\mathcal{D}_t$ is chosen in a way that for all $X \subseteq E$, the discrepancy between the relative frequency of X in $\mathcal{S}_t$ and that in $\mathcal{D}_t$ is at most ϵ with probability at least $1 - \delta$, i.e., s satisfies

$$\Pr\left(\left|\frac{|\mathcal{S}_t[X]|}{t} - \frac{|\mathcal{D}_t[X]|}{s}\right| \leq \epsilon\right) \geq 1 - \delta \tag{2}$$

for any $X \subseteq E$. The parameters ϵ (*error*) and δ (*confidence*) are specified by the user. Our extensive experiments in Sect. 5.2 show that a very close approximation of the true family of $\tilde{\Delta}$-closed itemsets can be obtained in this way.

Our algorithm recalculates the family of $\tilde{\Delta}$-closed itemsets *not* after each new transaction, but either upon request or after b new transactions have been received since the last update, where b, the *buffer size*, is specified by the user. Given $\mathcal{S}_t = \langle T_1, \ldots, T_t \rangle$ and $\mathcal{S}_{t'} = \langle T_1, \ldots, T_t, T_{t+1}, \ldots, T_{t'} \rangle$ with $t' - t \leq b$, the new sample $\mathcal{D}_{t'}$ of $\mathcal{S}_{t'}$ is computed from the old sample $\mathcal{D}_t$ by $\mathcal{D}_{t'} = \mathcal{D}_t \ominus \mathcal{D}_{\text{del}} \oplus \mathcal{D}_{\text{ins}}$, where $\mathcal{D}_{\text{del}}$ (resp. $\mathcal{D}_{\text{ins}}$) is the multiset of transactions to be removed from (resp. added to) $\mathcal{D}_t$, and $\ominus$ and $\oplus$ denote the set difference and the union operations on multisets.

The algorithm will be illustrated on the example transactional data stream given in Fig. 1 with $b = 8$ and $\Delta = 2$. For the sake of simplicity, we assume that transaction 1 will be replaced with transaction 9 by the sampling algorithm. The strongly closed itemsets for the first respectively last eight transactions are shown in Figures 2 and 3.

Fig. 2 2-closed itemsets for transactions 1–8

itemset	support
d	7
ad	2
bd	3
cd	2

Fig. 3 2-closed itemsets for transactions 2–9

itemset	support
c	4
d	6
ad	2
bd	2
cd	2

The rest of this section is organized as follows. We sketch the sampling algorithm in Sect. 4.1 and describe the algorithm updating the family of $\tilde{\Delta}$-closed itemsets from $\mathcal{D}_t$ to $\mathcal{D}_{t'}$ in Sect. 4.2.

4.1 Sampling

We use *reservoir sampling* (Knuth 1997; Vitter 1985) for generating a random sample $\mathcal{D}_t$ of size s for a data stream $\mathcal{S}_t = \langle T_1, \ldots, T_t \rangle$, as this method does not require the stream length to be known in advance. The general scheme of reservoir algorithms is that they first add $T_1, \ldots, T_s$ to a "reservoir" and then throw a biased coin with probability s/k of head for all $k = s+1, \ldots, t$. If the outcome is head they replace one of the elements selected from the reservoir uniformly at random with T_k. This naive version of reservoir sampling, attributed to A.G. Waterman by D. Knuth in Knuth (1997), generates a random sample $\mathcal{D}_t$ of $\mathcal{S}_t$ without replacement uniformly at random. That is, all elements of $\mathcal{S}_t$ have probability s/t of being part of the sample after $\mathcal{S}_t$ has been processed. We have implemented Vitter's more sophisticated version, called Algorithm Z in Vitter (1985).

Given a sample $\mathcal{D}_t$ of a data stream $\mathcal{S}_t = \langle T_1, \ldots, T_t \rangle$, the sample $\mathcal{D}_{t'}$ for $\mathcal{S}_{t'} = \langle T_1, \ldots, T_t, T_{t+1}, \ldots, T_{t'} \rangle$ is computed from $\mathcal{D}_t$ by repeatedly applying Algorithm Z to $\mathcal{D}_t$ and the elements in $\langle T_{t+1}, \ldots, T_{t'} \rangle$. Recall that $t' - t \leq b$, where b is the buffer size. If a transaction in the sample is replaced by a new transaction $T \in \{T_{t+1}, \ldots, T_{t'}\}$, we appropriately update a database $\mathcal{D}_{\text{del}}$ containing the transactions to be removed from $\mathcal{D}_t$ and a database $\mathcal{D}_{\text{ins}}$ containing the transactions to be added to $\mathcal{D}_t$. Clearly, $|\mathcal{D}_{\text{del}}| = |\mathcal{D}_{\text{ins}}|$. Furthermore,

$$\mathbb{E}[|\mathcal{D}_{\text{del}}|] = \mathbb{E}[|\mathcal{D}_{\text{ins}}|] \leq \frac{bs}{t'} \ . \tag{3}$$

This follows directly from the linearity of the expectation and from $\mathbb{E}[X_k] = s/t'$, where X_k is the indicator random variable for the event that T_k is selected for $\mathcal{S}_{t'}$. Note that in (3) we have inequality only in the case that $t' - t < b$, i.e., when an update is calculated upon request for an incomplete buffer; o/w we always have equality. The RHS of (3) approaches 0 as t' approaches infinity. For example, it is only 15 for $b = 10\text{k}$, $t' = 100\text{M}$, $\epsilon = 0.005$, $\delta = 0.001$, and $s = 150\text{k}$, where the sample size $s = s(\epsilon, \delta)$ satisfying (2) is calculated by Hoeffding's inequality,[4] i.e.,

[4] We note that Hoeffding's inequality applies to samples without replacement as well (Hoeffding 1963). A tighter bound can be derived from Serfling's inequality (Serfling 1974). The improvement becomes however marginal with increasing data stream length.

$$s = \left\lceil \frac{1}{2\epsilon^2} \ln \frac{2}{\delta} \right\rceil \quad . \tag{4}$$

4.2 Incremental update

Note that the sample size s in (4) depends on the error and confidence parameters ϵ and δ only. That is, s does not change with increasing data stream length. Hence, both denominators in the LHS of (1) will be fixed (i.e., s) for the entire mining process from the s-th transaction onward. More precisely, for any transaction database $\mathcal{D}$ of size s and $\tilde{\Delta} \in [0, 1]$, the family of *relatively* $\tilde{\Delta}$-closed itemsets of $\mathcal{D}$ is equal to the family $\mathcal{C}_{\Delta,\mathcal{D}}$ of *absolutely* Δ-closed itemsets for $\Delta = \lceil s\tilde{\Delta} \rceil$. This allows us to consider the following problem equivalent to the $\tilde{\Delta}$-Closed Set Listing problem:

> **Δ-Closed Set Listing Problem** *Given* $\mathcal{D}_t$, $\mathcal{D}_{\text{del}}$, $\mathcal{D}_{\text{ins}}$ for $\mathcal{S}_t$ and $\mathcal{S}_{t'}$ as defined in Sect. 4.1, an integer $\Delta > 0$, and the family $\mathcal{C}_{\Delta,\mathcal{D}_t}$ of Δ-closed itemsets of $\mathcal{D}_t$, *generate* all elements of $\mathcal{C}_{\Delta,\mathcal{D}_{t'}}$ for $\mathcal{D}_{t'} = \mathcal{D}_t \ominus \mathcal{D}_{\text{del}} \oplus \mathcal{D}_{\text{ins}}$.

Instead of generating $\mathcal{C}_{\Delta,\mathcal{D}_{t'}}$ from scratch, our goal is to design a much faster *practical* algorithm by reducing the number of evaluations of the closure operator for $\mathcal{D}_{t'}$. This is motivated by the fact that the execution of the closure operator is the most expensive part of the algorithm. We make use of the fact that the expected number of changes in $\mathcal{D}_{t'}$ w.r.t. $\mathcal{D}_t$ becomes smaller and smaller as t' increases (cf. (3)). Accordingly, our focus in the design of the updating algorithm is on *quickly* deciding whether an element $C' \in \mathcal{C}_{\Delta,\mathcal{D}_t}$ remains Δ-closed in $\mathcal{D}_{t'}$, where C' is obtained by $C' = \sigma_{\Delta,\mathcal{D}_t}(C \cup \{e\})$ for some $C \in \mathcal{C}_{\Delta,\mathcal{D}_t}$ and $e \in E$. Below we show that in all of the cases when at least one of the support sets $\mathcal{D}_{\text{del}}[C \cup \{e\}]$ or $\mathcal{D}_{\text{ins}}[C \cup \{e\}]$ is empty, the problem above can be decided much faster than with the naive way of using Algorithm 1. As we empirically demonstrate in Sect. 5, a considerable speed-up over the naive algorithm can be achieved in this way.

We first briefly sketch the algorithm computing $\mathcal{C}_{\Delta,\mathcal{D}_{t'}}$ from $\mathcal{C}_{\Delta,\mathcal{D}_t}$ (see Algorithm 2). It requires four auxiliary pieces of information for all strongly closed itemsets in $\mathcal{C}_{\Delta,\mathcal{D}_t}$, except for the empty set (cf. Line 1 of Main). Hence, to simplify the notation, the set variables $\mathcal{C}_{\Delta,\mathcal{D}_t}$ and $\mathcal{C}_{\Delta,\mathcal{D}_{t'}}$ in all algorithms of this section store quintuples, where the first component is the strongly closed itemset itself; the other four components are specified below.

Algorithm 2 is a divide and conquer algorithm that recursively calls ListClosed with some Δ-closed set $C \in \mathcal{C}_{\Delta,\mathcal{D}_{t'}}$, forbidden set $N \subseteq E$, and minimum candidate generator element i. It first determines the next smallest generator element e (Line 3) and calculates the closure $C' = \sigma_{\Delta,\mathcal{D}_{t'}}(C \cup \{e\})$ in Lines 4–10; these steps are discussed in detail below. We store C', together with some auxiliary information (Lines 12 and 15). The algorithm then calls ListClosed recursively for generating further Δ-closed supersets of C'. In particular, if C' does not contain any forbidden item from N then the last element of the quintuple stored for C' is $\uparrow$ (Line 12); o/w it is $\downarrow$ (Line 15). After all elements of $\mathcal{C}_{\Delta,\mathcal{D}_{t'}}$ have been generated that are supersets of C, contain e, but do not contain any element in N, the algorithm generates all closed sets in $\mathcal{C}_{\Delta,\mathcal{D}_{t'}}$ that are supersets of C and do not contain any element from $N \cup \{e\}$ (Lines 16–19).

Example 1 Using the transactions in Fig. 1, we show how Algorithm 2 updates the family of 2-closed itemsets for the first eight transactions (cf. Fig. 2) to that for the last eight (cf. Fig. 3). For $E = \{a, b, c, d\}$ with $a < b < c < d$ and $\mathcal{C}_{\Delta,\mathcal{D}_t} = \{d, ad, bd, cd\}$, the input to the algorithm for this update consists of $\mathcal{D}_{\text{del}} = \{t_1\}$, $\mathcal{D}_{\text{ins}} = \{t_9\}$, and $\Delta = 2$. The algorithm first initializes $\mathcal{C}_{\Delta,\mathcal{D}_{t'}} \leftarrow \{\emptyset\}$ (line 2) and then calls ListClosed($\emptyset$, $\emptyset$, a) (line 3). The recursive

Algorithm 2 UPDATE $\mathcal{C}_{\Delta,\mathcal{D}_t}$

input: datasets $\mathcal{D}_{\text{del}}, \mathcal{D}_{\text{ins}}$ over E and $\Delta \in \mathbb{N}$
require: totally ordered set $(E, \preceq)$, dataset $\mathcal{D}_t$ over E, and $\mathcal{C}_{\Delta,\mathcal{D}_t}$
output: $\mathcal{C}_{\Delta,\mathcal{D}_{t'}}$ for $\mathcal{D}_{t'} = \mathcal{D}_t \ominus \mathcal{D}_{\text{del}} \oplus \mathcal{D}_{\text{ins}}$

MAIN:
1: $\mathcal{C}_{\Delta,\mathcal{D}_{t'}} \leftarrow \{\emptyset\}$
2: LISTCLOSED($\emptyset, \emptyset, \min E$)

LISTCLOSED(C, N, i):
1: $X \leftarrow \{k \in E \setminus C : k \geq i\}$
2: **if** $X \neq \emptyset$ **then**
3: $e \leftarrow \min X$; $C_e \leftarrow C \cup \{e\}$
4: **if** $\mathcal{D}_{\text{del}}[C_e] = \emptyset \wedge \mathcal{D}_{\text{ins}}[C_e] = \emptyset$ **then**
5: $C' \leftarrow$ CLOSURE_$\alpha(C, e, \mathcal{C}_{\Delta,\mathcal{D}_t})$ ▷ Case (α)
6: **else if** $\mathcal{D}_{\text{ins}}[C_e] = \emptyset$ **then**
7: $C' \leftarrow$ CLOSURE_$\beta(C, e, \mathcal{D}_{\text{del}}[C_e], \mathcal{C}_{\Delta,\mathcal{D}_t})$ ▷ Case (β)
8: **else if** $\mathcal{D}_{\text{del}}[C_e] = \emptyset$ **then**
9: $C' \leftarrow$ CLOSURE_$\gamma(C, e, \mathcal{D}_{\text{ins}}[C_e], \mathcal{C}_{\Delta,\mathcal{D}_t})$ ▷ Case (γ)
10: **else** $C' \leftarrow \sigma_{\Delta,\mathcal{D}_{t'}}(C_e)$ ▷ Case (δ)
11: **if** $C' \cap N = \emptyset$ **then**
12: add $(C, e, N, C', \uparrow)$ to $\mathcal{C}_{\Delta,\mathcal{D}_{t'}}$
13: LISTCLOSED($C', N, e + 1$)
14: **else**
15: add $(C, e, N, C', \downarrow)$ to $\mathcal{C}_{\Delta,\mathcal{D}_{t'}}$
16: $Y \leftarrow \{k \in E \setminus C : k > e\}$
17: **if** $Y \neq \emptyset$ **then**
18: $e' \leftarrow \min Y$
19: LISTCLOSED($C, N \cup \{e\}, e'$)

calls of list closed are visualized in Fig. 4. The edges corresponding to lines 1–15 are labeled with the value of the variable C_e (cf. line 3) and the case used for update in lines 4–10; unlabeled edges correspond to lines 17–19.

Theorem 2 *Algorithm 2 generates all elements of $\mathcal{C}_{\Delta,\mathcal{D}_{t'}}$ correctly, irredundantly, in total time $O\left(|E| \cdot |\mathcal{C}_{\Delta,\mathcal{D}_{t'}}| \cdot \|\mathcal{D}_{t'}\|_0\right)$, with delay $O\left(|E|^2 \|\mathcal{D}_{t'}\|_0\right)$, and in space $O\left(|E| + \|\mathcal{D}_{t'}\|_0\right)$.*

Proof Regarding the correctness, we only need to show that C' computed in Lines 4–10 satisfies $C' = \sigma_{\Delta,\mathcal{D}_{t'}}(C \cup \{e\})$. The correctness of CLOSURE_α (Algorithm 3), CLOSURE_β (Algorithm 4), and CLOSURE_γ (Algorithm 5) is shown below in Lemmas 1, 2, and 3, respectively. The proofs of the irredundancy and the time and space complexity are immediate from Boley et al. (2010) and Gély (2005) by noting that Algorithm 2 must call the closure operator for all elements in $\mathcal{C}_{\Delta,\mathcal{D}_{t'}}$ in the worst case. □

In the rest of this section we give the algorithms for the cases distinguished in Lines 4–10 (case (δ) is trivial) and prove their correctness.
Case (α) We first consider the case that the set $C \cup \{e\}$ with $C \in \mathcal{C}_{\Delta,\mathcal{D}_{t'}}$ and $e \in E$ to be extended for further Δ-closed sets satisfies

$$\mathcal{D}_{\text{del}}[C \cup \{e\}] = \emptyset \text{ and } \mathcal{D}_{\text{ins}}[C \cup \{e\}] = \emptyset \tag{5}$$

(Lines 4–5 of Algorithm 2). The closure $\sigma_{\Delta,\mathcal{D}_{t'}}(C \cup \{e\})$ for this case can be computed by Algorithm 3; the correctness of Algorithm 3 is stated in Lemma 1 below.

Algorithm 3 CLOSURE_α

input: $C \in \mathcal{C}_{\Delta,\mathcal{D}_{t'}}$ with $\mathcal{D}_{t'} = \mathcal{D}_t \ominus \mathcal{D}_{\text{del}} \oplus \mathcal{D}_{\text{ins}}$, $e \in E$, and $\mathcal{C}_{\Delta,\mathcal{D}_t}$
require: $\mathcal{D}_t$
output: $\sigma_{\Delta,\mathcal{D}_{t'}}(C \cup \{e\})$

1: **if** $\exists (C, e, N, C', q) \in \mathcal{C}_{\Delta,\mathcal{D}_t}$ for some N, C', and q **then return** C'
2: **else return** $\sigma_{\Delta,\mathcal{D}_t}(C \cup \{e\})$ ▷ $\sigma_{\Delta,\mathcal{D}_t}(C \cup \{e\}) = \sigma_{\Delta,\mathcal{D}_{t'}}(C \cup \{e\})$ for this case

Lemma 1 *Algorithm 3 is correct, i.e., for all $C \in \mathcal{C}_{\Delta,\mathcal{D}_{t'}}$ and for all $e \in E$, the output of the algorithm is $\sigma_{\Delta,\mathcal{D}_{t'}}(C \cup \{e\})$.*

Proof Condition (5) implies that $\mathcal{D}_t[C \cup \{e\}] = \mathcal{D}_{t'}[C \cup \{e\}]$, where $\mathcal{D}_{t'} = \mathcal{D}_t \ominus \mathcal{D}_{\text{del}} \oplus \mathcal{D}_{\text{ins}}$. Hence, $\sigma_{\Delta,\mathcal{D}_{t'}}(C \cup \{e\}) = \sigma_{\Delta,\mathcal{D}_t}(C \cup \{e\})$ and $\sigma_{\Delta,\mathcal{D}_{t'}}(C \cup \{e\}) \in \mathcal{C}_{\Delta,\mathcal{D}_t}$, from which the proof is immediate for both cases considered in Lines 1–2. □

Example 2 In our running Example 1, the first call LC(∅,∅,a) in Fig. 4 corresponds to case (α), as $\mathcal{D}_{\text{ins}}[a] = \mathcal{D}_{\text{del}}[a] = \emptyset$. Algorithm 3 returns ad as the closure of a in line 1, i.e., we do not need to (re)evaluate the closure operator on a.

Case (β) We now turn to the case that $C \in \mathcal{C}_{\Delta,\mathcal{D}_{t'}}$ and $e \in E$ fulfill

$$\mathcal{D}_{\text{del}}[C \cup \{e\}] \neq \emptyset \text{ and } \mathcal{D}_{\text{ins}}[C \cup \{e\}] = \emptyset \tag{6}$$

(Lines 6–7 of Algorithm 2). In Proposition 1 below we first prove some monotonicity results that will be used also for case (γ).

Proposition 1 *Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be transaction databases over E. If $\mathcal{D}_1 \subseteq \mathcal{D}_2$ then for all $\Delta \in \mathbb{N}$,*

$$\mathcal{C}_{\Delta,\mathcal{D}_1} \subseteq \mathcal{C}_{\Delta,\mathcal{D}_2} \ . \tag{7}$$

Furthermore, for all $\Delta \in \mathbb{N}$ and for all $X \subseteq E$,

$$\sigma_{\Delta,\mathcal{D}_1}(X) \supseteq \sigma_{\Delta,\mathcal{D}_2}(X) \ . \tag{8}$$

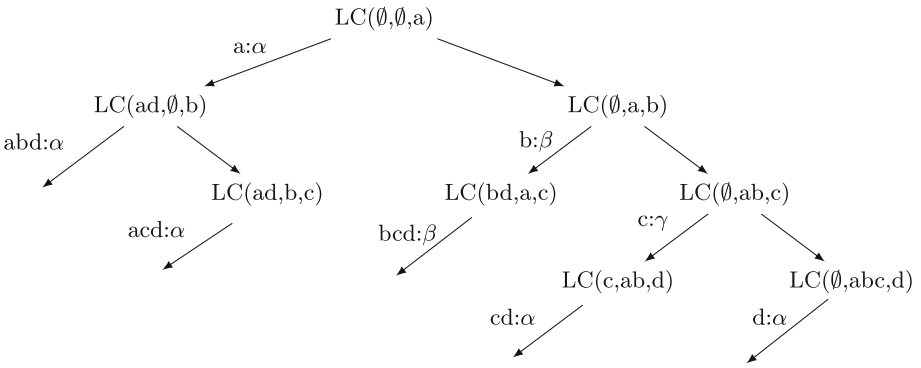

Fig. 4 Call stack for the update in Example 1. Labeled edges correspond to lines 1–15 of Algorithm 2; unlabeled to lines 17–19. The first component of an edge label denotes C_e (cf. line 3 of Algorithm 2), the second the case applied (cf. lines 4–10)

Algorithm 4 CLOSURE_β

input: $C \in \mathcal{C}_{\Delta,\mathcal{D}_{t'}}$ with $\mathcal{D}_{t'} = \mathcal{D}_t \ominus \mathcal{D}_{\text{del}} \oplus \mathcal{D}_{\text{ins}}$, $e \in E$, $\mathcal{D}_{\text{del}}[C \cup \{e\}]$, and $\mathcal{C}_{\Delta,\mathcal{D}_t}$
require: $\mathcal{D}_t$
output: $\sigma_{\Delta,\mathcal{D}_{t'}}(C \cup \{e\})$

1: **if** there exists (C, e, N, C', q) in $\mathcal{C}_{\Delta,\mathcal{D}_t}$ for some N, C', and q **then**
2: $\quad C'.\text{count} \leftarrow |\mathcal{D}_t[C']| - |\mathcal{D}_{\text{del}}[C']|$
3: $\quad$ **for all** $i \in E \setminus C'$ **do**
4: $\quad\quad C'.\Delta_i \leftarrow |\mathcal{D}_t[C' \cup \{i\}]|$
5: $\quad\quad$ **if** $C'.\text{count} - C'.\Delta_i + |\mathcal{D}_{\text{del}}[C' \cup \{i\}]| < \Delta$ **then**
6: $\quad\quad\quad$ **return** $\sigma_{\Delta,\mathcal{D}_{t'}}(C \cup \{e\})$
7: $\quad$ **return** C'
8: **else**
9: $\quad$ **return** $\sigma_{\Delta,\mathcal{D}_{t'}}(C \cup \{e\})$

Proof Let $C \in \mathcal{C}_{\Delta,\mathcal{D}_1}$ for some $\Delta \in \mathbb{N}$ and let $\mathcal{D}' = \mathcal{D}_2 \ominus \mathcal{D}_1$. Then, for any $e \in E \setminus C$, we have

$$\begin{aligned} |\mathcal{D}_2[C \cup \{e\}]| &= |\mathcal{D}_1[C \cup \{e\}]| + |D'[C \cup \{e\}]| \\ &\leq |\mathcal{D}_1[C]| - \Delta + |D'[C]| \\ &= |\mathcal{D}_2[C]| - \Delta \ , \end{aligned}$$

where the inequality follows from $C \in \mathcal{C}_{\Delta,\mathcal{D}_1}$ and from the anti-monotonicity of support sets. Hence $C \in \mathcal{C}_{\Delta,\mathcal{D}_2}$ completing the proof of (7).

To show (8), suppose that during the calculation of $\sigma_{\Delta,\mathcal{D}_2}(X)$, the items in $\sigma_{\Delta,\mathcal{D}_2}(X) \setminus X$ have been added to X in the order $e_1, \ldots, e_k$. Let $X_0 = X$ and $X_i = X \cup \{e_1, \ldots, e_{i-1}, e_i\}$ for all $i \in [k]$. Then $|\mathcal{D}_2[X_{i-1}]| - |\mathcal{D}_2[X_i]| < \Delta$ for all $i \in [k]$ (see Algorithm 1). Since $\mathcal{D}_2[X_{i-1}] \supseteq \mathcal{D}_2[X_i]$ and $\mathcal{D}_1 \subseteq \mathcal{D}_2$, we have $|\mathcal{D}_1[X_{i-1}]| - |\mathcal{D}_1[X_i]| < \Delta$ for all i. Thus, as Algorithm 1 is Church-Rosser, all e_i will be added to $\sigma_{\Delta,\mathcal{D}_1}(X)$ as well, implying (8). □

Using Proposition 1, we have the following result for Algorithm 4 concerning case (β):

Lemma 2 *Algorithm 4 is correct, i.e., for all $C \in \mathcal{C}_{\Delta,\mathcal{D}_{t'}}$ and for all $e \in E$, the output of the algorithm is $\sigma_{\Delta,\mathcal{D}_{t'}}(C \cup \{e\})$.*

Proof By Condition (6), $\mathcal{D}_{t'}[C \cup \{e\}] \subseteq \mathcal{D}_t[C \cup \{e\}]$ and hence Proposition 1 implies that there is no $Y \in \mathcal{C}_{\Delta,\mathcal{D}_{t'}}$ with $C \cup \{e\} \subsetneq Y \subsetneq \sigma_{\Delta,\mathcal{D}_t}(C \cup \{e\})$. Furthermore, if $\sigma_{\Delta,\mathcal{D}_t}(C \cup \{e\}) \notin \mathcal{C}_{\Delta,\mathcal{D}_{t'}}$ then $\sigma_{\Delta,\mathcal{D}_t}(C \cup \{e\}) \subsetneq \sigma_{\Delta,\mathcal{D}_{t'}}(C \cup \{e\})$. Thus, to check whether $C' = \sigma_{\Delta,\mathcal{D}_t}(C \cup \{e\})$ remains closed in $\mathcal{D}_{t'}$, it suffices to test whether

$$|\mathcal{D}_{t'}[C']| - |\mathcal{D}_{t'}[C' \cup \{i\}]| \geq \Delta \tag{9}$$

further holds for all items $i \in E \setminus C'$ (Lines 2–6 of Algorithm 4). If so, the algorithm returns C' in Line 7, implying the correctness of Algorithm 4 for the case that $C' \in \mathcal{C}_{\Delta,\mathcal{D}_{t'}}$; the claim is trivial for the other two cases (Lines 6 and 9). □

We note that in our implementation of Algorithm 4 we do not calculate $C'.\text{count}$ and $C'.\Delta_i$ in Lines 2 and 4, but store and maintain them consistently. In this way, the condition in Line 5 can be decided from $\mathcal{D}_{\text{del}}$, without any access to $\mathcal{D}_t$. It is important to mention that with increasing stream length, the number of elements to be deleted from $\mathcal{C}_{\Delta,\mathcal{D}_t}$ becomes smaller (cf. (3)) and typically, most of the elements of $\mathcal{C}_{\Delta,\mathcal{D}_{t'}}$ are calculated by terminating in Line 7.

Algorithm 5 CLOSURE_γ

input: $C \in \mathcal{C}_{\Delta,\mathcal{D}_{t'}}$ with $\mathcal{D}_{t'} = \mathcal{D}_t \ominus \mathcal{D}_{\text{del}} \oplus \mathcal{D}_{\text{ins}}$, $e \in E$, $\mathcal{D}_{\text{ins}}[C \cup \{e\}]$, and $\mathcal{C}_{\Delta,\mathcal{D}_t}$
require: $\mathcal{D}_t$
output: $\sigma_{\Delta,\mathcal{D}_{t'}}(C \cup \{e\})$

1: **if** there exists (C, e, N, C', q) in $\mathcal{C}_{\Delta,\mathcal{D}_t}$ for some N, C', and q **then**
2: $C'' \leftarrow C \cup \{e\}$; $\mathcal{D}' \leftarrow (\mathcal{D}_t \oplus \mathcal{D}_{\text{ins}})[C'']$
3: **repeat**
4: **for all** $i \in C' \setminus C''$ **do**
5: **if** $|\mathcal{D}'| - |\mathcal{D}'[i]| < \Delta$ **then**
6: $C'' \leftarrow C'' \cup \{i\}$; $\mathcal{D}' \leftarrow \mathcal{D}'[i]$
7: **until** $\mathcal{D}'$ has not been changed in Loop 4–6
8: **return** C''
9: **else**
10: **return** $\sigma_{\Delta,\mathcal{D}_{t'}}(C \cup \{e\})$

Example 3 In our running Example 1, the call of LC(∅,*a*,*b*) in Fig. 4 corresponds to case (β) because $\mathcal{D}_{\text{ins}}[b] = \emptyset$ and $\mathcal{D}_{\text{del}}[b] \neq \emptyset$. Since $(\emptyset, a, b, bd, \uparrow) \in \mathcal{C}_{\Delta,\mathcal{D}_t}$, Algorithm 4 only needs to compute support queries on $\mathcal{D}_{\text{del}}$ in lines 2, 4 and 5. For all i considered in line 3, the condition in line 5 is not fulfilled. Hence, the algorithm returns db in line 7, without calling the closure operator.

Case (γ) Finally we discuss the case that $C \in \mathcal{C}_{\Delta,\mathcal{D}_{t'}}$ and $e \in E$ satisfy the condition

$$\mathcal{D}_{\text{del}}[C \cup \{e\}] = \emptyset \text{ and } \mathcal{D}_{\text{ins}}[C \cup \{e\}] \neq \emptyset \tag{10}$$

(see Lines 8–9 of Algorithm 2). The proof for this case is shown also by using Proposition 1.

Lemma 3 *Algorithm 5 is correct, i.e., for all $C \in \mathcal{C}_{\Delta,\mathcal{D}_{t'}}$ and for all $e \in E$, the output of the algorithm is $\sigma_{\Delta,\mathcal{D}_{t'}}(C \cup \{e\})$.*

Proof The proof is automatic for the case that the condition in Line 1 of Algorithm 5 is false. Consider the case that it is true. Proposition 1 with Condition (10) implies that $\mathcal{C}_{\Delta,\mathcal{D}_t} \subseteq \mathcal{C}_{\Delta,\mathcal{D}_{t'}}$ (i.e., all Δ-closed itemsets in $\mathcal{C}_{\Delta,\mathcal{D}_t}$ are preserved) and that $\sigma_{\Delta,\mathcal{D}_{t'}}(C \cup \{e\}) \subseteq \sigma_{\Delta,\mathcal{D}_t}(C \cup \{e\})$. Thus, when calculating $\sigma_{\Delta,\mathcal{D}_{t'}}(C \cup \{e\})$ in Loop 3–7, it suffices to consider only the elements in $\sigma_{\Delta,\mathcal{D}_t}(C \cup \{e\}) \setminus (C \cup \{e\})$, from which the claim is immediate for this case. □

Compared to case (β), we need to calculate support counts in the entire sample $\mathcal{D}_{t'}$ for this case. However, the inner loop (Lines 4–6) iterates over a typically much smaller set than the general closure algorithm (cf. Lines 2–5 of Algorithm 1). Analogously to case (β), the number of new Δ-closed itemsets to be added to $\mathcal{C}_{\Delta,\mathcal{D}_{t'}}$ becomes smaller with increasing stream length, and hence, most of the elements of $\mathcal{C}_{\Delta,\mathcal{D}_{t'}}$ are calculated in the "then" part (Line 2–8) of the "if" statement.

Example 4 In our running Example 1, the call of LC(∅,*ab*,*c*) in Fig. 4 corresponds to case (γ) since item c occurs only in $\mathcal{D}_{\text{ins}}$ (i.e., $\mathcal{D}_{\text{ins}}[c] \neq \emptyset$ and $\mathcal{D}_{\text{del}}[c] = \emptyset$). Since $(\emptyset, ab, c, cd, \uparrow) \in \mathcal{C}_{\Delta,\mathcal{D}_t}$, the algorithm goes into the loop 4–6, iterating over all elements of $cd \setminus c$. The condition in line 5 is not satisfied for d and thus c is returned as a new closed itemset in line 8, without calling the closure operator.

4.2.1 Controlling the time and space complexity of the update

Although by Theorem 2 Algorithm 2 does not improve the worst-case time and space complexity of the batch algorithm (Boley et al. 2009) calculating the family of strongly closed sets from scratch, our experimental results presented in Sect. 5 clearly demonstrate a considerable speed-up on artificial and real-word datasets. The total time depends on the cardinality of $\mathcal{C}_{\Delta,\mathcal{D}_{t'}}$, which can be exponential in $|E|$. The time and space of the update can be controlled by selecting the parameter Δ in a way that $|\mathcal{C}_{\Delta,\mathcal{D}_{t'}}| < K$ for some reasonable small K. Once K has been fixed, the value of Δ can automatically be adjusted when the number of elements in $\mathcal{C}_{\Delta,\mathcal{D}_{t'}}$ that have already been enumerated exceeds K. More precisely, suppose Algorithm 2 has generated a subset $\mathcal{C}' \subseteq \mathcal{C}_{\Delta,\mathcal{D}_{t'}}$ with $|\mathcal{C}'| = K + 1$. For all $C \in \mathcal{C}'$, let Δ_C be the strength of C and denote $\Delta' = \min_{C\in\mathcal{C}'} \Delta_C$. Clearly, $\Delta' \geq \Delta$. Let $\mathcal{C}'' = \{C \in \mathcal{C}' : \Delta_C > \Delta'\}$. For the set obtained we have $\mathcal{C}'' \subseteq \mathcal{C}_{\Delta'+1,\mathcal{D}_{t'}}$ and $|\mathcal{C}''| \leq K$.

This change of Δ to $\Delta' + 1$ requires, however, the maintenance of auxiliary pieces of information for all already generated strongly closed sets, as well as the reconstruction of the five tuples for the closed sets remaining. More precisely, suppose $\Delta_{C,e} = |\mathcal{D}_t[C]| - |\mathcal{D}_t[C \cup \{e\}]|$ has been calculated correctly for all $C \in \mathcal{C}_{\Delta,\mathcal{D}_t}$ and for all $e \in E \setminus C$. Notice that the strength of C in $\mathcal{D}_t$ is given by $\min_{e\in E\setminus C} \Delta_{C,e}$, where the $\Delta_{C,e}$s are obtained as a byproduct of the algorithm computing the closure operator (cf. Algorithm 1). One can see that if $C \in \mathcal{C}_{\Delta,\mathcal{D}_{t'}}$ and C has not been recalculated by calling the closure operator, then $\Delta_{C,e}$ can be updated by

$$\Delta_{C,e} = \Delta_{C,e} + |\mathcal{D}_{\text{ins}}[C]| - |\mathcal{D}_{\text{ins}}[C \cup \{e\}]| - |\mathcal{D}_{\text{del}}[C]| + |\mathcal{D}_{\text{del}}[C \cup \{e\}]|$$

for all $e \in E \setminus C$. Thus, the complexity of the update for this case depends on the cardinality of $\mathcal{D}_{\text{ins}}$ and $\mathcal{D}_{\text{del}}$ only, which become smaller and smaller with increasing t' by (3). Finally, utilizing the algebraic properties of closure systems, the five tuples can be reconstructed by a top-down traversal of the enumeration tree corresponding to Algorithm 2.

5 Empirical evaluation

In this section we empirically evaluate our algorithm on artificial and real-world datasets. In particular, we experimentally demonstrate that it results in a considerable speed-up (Sect. 5.1) and has high approximation quality (Sect. 5.2). The data streams, all consisting of 5M transactions, were generated from benchmark datasets from the UCI Machine Learning (Dua and Graff 2019) and from the Frequent Itemset Mining Dataset[5] repositories (see Table 1). For each dataset $\mathcal{D}$, Table 1 contains the cardinality of the ground set ($|E|$), the number of transactions ($|\mathcal{D}|$), and the density defined by $\frac{1}{|E|\cdot|\mathcal{D}|} \sum_{T\in\mathcal{D}} |T|$.

5.1 Speed-up

In this section we empirically study the speed-up obtained by our algorithm. For this purpose, we first sample 100k random transactions, replace then 10k, 1k, 100, 10, and 1 transaction in the sample, and (i) run our algorithm as well as (ii) update the sample and run the algorithm that corresponds to Algorithm 2 with $\mathcal{C}_{\Delta,\mathcal{D}_t} = \emptyset$. Notice that (ii) corresponds to a classical batch computation. Henceforth we refer to the algorithm for (ii) as the *batch* algorithm. We

[5] http://fimi.ua.ac.be/data/.

Table 1 Benchmark datasets used in the experiments

Name	Kosarak	Mushroom	Poker-hand	Retail	T10I4D100K	T40I10D100K
$\|E\|$	41,270	119	95	16,470	870	942
$\|\mathcal{D}\|$	990,002	8124	1,025,010	88,162	100,000	100,000
Density	0.000196	0.193277	0.115789	0.000626	0.011612	0.042044

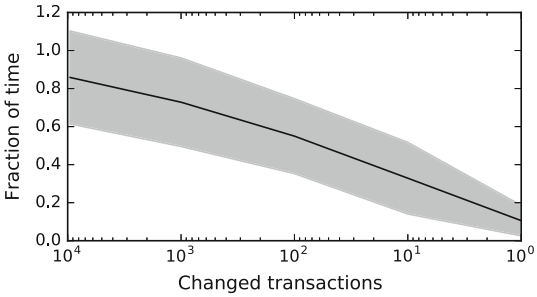

Fig. 5 Fraction of the runtime of our streaming and the batch algorithm as a function of the number of changes (log scale): black: mean, gray: SD

Fig. 6 Runtime in seconds of our streaming and the batch algorithm obtained for T10I4D100k for different number of changes and for $\tilde{\Delta} = 0.006$

#Changes	Stream time	Batch time
10,000	6.0	6.0
1,000	4.7	6.0
100	4.2	6.0
10	1.1	6.0
1	0.3	6.0

define the speed-up, denoted S, by the runtime of the batch algorithm over that of the streaming algorithm. Figure 5 shows the average runtime fraction of our algorithm in comparison to the batch algorithm as a function of the number of changed transactions for all datasets from Table 1. The runtime results are reported in detail for one dataset in Fig. 6 by noting that we observed a similar speed-up for all other datasets. As the number of changes decreases, our streaming algorithm needs to evaluate considerably less database queries, implying that the smaller the change in the sample, the more the runtime of the two settings differs. In Table 2 we present the number of strongly closed itemsets ($|\mathcal{C}_{\Delta,\mathcal{D}_t}|$) and the speed-up ($S$) of our algorithm for various values of $\tilde{\Delta}$ for such experiments when only a single transaction has been changed in the sample. In most of the cases our algorithm is faster by at least one order of magnitude. One can also observe that the more $\tilde{\Delta}$-closed itemsets are calculated (or equivalently, the smaller $\tilde{\Delta}$ is) , the higher the speed-up. This is not surprising, as the batch algorithm needs to call the closure operator for *all* strongly closed sets, whereas our algorithm only for a subset of them. Recall that a transaction is added to the sample with probability s/k, where s is the size of the sample and k the current length of the data stream. The expected value to replace only one transaction is reached if the probability to replace each of the s transactions in the sample is at most $1/s$. This condition holds, whenever $k \geq s^2$.

5.2 Approximation quality

In this section we present empirical results demonstrating the high approximation quality of our algorithm measured in terms of precision and recall. For these experiments, we use

Table 2 Number of $\tilde{\Delta}$-closed sets and speed-up (S) for changing a single transaction

$\tilde{\Delta}$	Kosarak		Mushroom		Poker-hand		Retail		T10I4D100K		T40I10D100K	
	$\lvert\mathcal{C}_{\Delta,\mathcal{D}_t}\rvert$	S	$\lvert\mathcal{C}_{\Delta,\mathcal{D}_t}\rvert$	S	$\lvert\mathcal{C}_{\Delta,\mathcal{D}_t}\rvert$	S	$\lvert\mathcal{C}_{\Delta,\mathcal{D}_t}\rvert$	S	$\lvert\mathcal{C}_{\Delta,\mathcal{D}_t}\rvert$	S	$\lvert\mathcal{C}_{\Delta,\mathcal{D}_t}\rvert$	S
0.046	8	3.09	154	10.90	62	8.26	8	5.69	6	7.69	191	16.57
0.041	8	5.81	186	13.74	62	11.55	10	2.89	11	8.00	222	16.52
0.036	9	5.48	245	5.89	127	11.33	11	8.05	19	11.48	267	18.73
0.031	10	7.20	385	7.24	247	14.74	12	3.67	29	9.89	330	14.37
0.026	14	7.36	547	18.04	353	13.17	13	7.69	44	26.15	433	22.79
0.021	15	7.31	1105	19.44	578	19.45	13	9.28	83	24.57	649	23.40
0.016	24	11.13	2012	23.44	738	36.43	18	11.00	138	16.77	1146	43.30
0.011	38	14.87	4367	34.73	739	26.43	26	12.58	219	21.77	2780	96.39
0.006	86	23.61	11k	33.96	4238	39.35	67	24.63	391	50.05	11k	235.86
0.001	1148	99.79	82k	37.41	46k	59.66	1638	106.19	2574	148.09	346k	1893

Table 3 Number of $\tilde{\Delta}$-closed sets ($|\mathcal{C}_{\Delta,\mathcal{D}_t}|$), precision ($P$) and recall ($R$) after processing 5M transactions for various datasets and different values of $\tilde{\Delta}$

$\tilde{\Delta}$	Kosarak			Mushroom			Poker-hand			Retail			T10I4D5M			T40I10D5M		
	$\lvert\mathcal{C}_{\Delta,\mathcal{D}_t}\rvert$	P	R	$\lvert\mathcal{C}_{\Delta,\mathcal{D}_t}\rvert$	P	R	$\lvert\mathcal{C}_{\Delta,\mathcal{D}_t}\rvert$	P	R	$\lvert\mathcal{C}_{\Delta,\mathcal{D}_t}\rvert$	P	R	$\lvert\mathcal{C}_{\Delta,\mathcal{D}_t}\rvert$	P	R	$\lvert\mathcal{C}_{\Delta,\mathcal{D}_t}\rvert$	P	R
0.046	8	1	1	155	0.99	1	6	1	1	8	1	1	6	1	1	190	1	0.99
0.041	8	1	1	190	1	0.99	62	1	1	10	1	1	11	0.92	1	223	0.98	0.99
0.036	9	1	1	225	0.98	0.98	127	1	1	11	1	1	19	1	1	267	0.99	0.99
0.031	10	1	1	382	0.99	1	248	1	1	12	1	1	29	1	0.93	330	0.99	0.99
0.026	14	1	1	676	0.97	0.98	353	1	1	13	1	1	44	0.96	0.98	433	1	0.98
0.021	16	1	0.94	1112	0.99	1	578	1	1	13	0.93	1	82	0.99	0.98	650	1	0.99
0.016	24	1	1	1934	0.97	1	738	1	1	18	1	1	140	1	0.99	1137	0.98	0.98
0.011	40	0.98	1	4361	0.84	0.84	739	1	1	27	1	0.96	218	0.98	0.99	2785	0.98	0.98
0.006	86	0.98	0.97	9469	0.80	0.93	4343	0.96	0.94	66	0.98	0.98	390	0.99	1	11k	0.97	0.97
0.001	1153	0.93	0.96	76k	0.93	0.98	47k	1	1	1653	0.93	0.96	2591	0.96	0.94	–	–	–

data streams of length 5M obtained by random enlargement of the benchmark datasets listed in Table 1, as well as 10 artificial data streams (T10I4D5M, T40I10D5M, and 8 variations of T10I4D5M), each of length 5M, generated with the IBM Quest data generator. This software generates synthetic market basket datasets based on user defined parameters. The parameters are average transaction size (T), average length of maximal patterns (I), number of transactions (D), number of patterns (L), correlation strength between patterns (C), and the number of different items in thousands (N). For the two artificial data streams (T10I4D5M and T40I10D5M) we used the same parameters (except for the size) as for T10I4D100K and T40I10D100K. For the variations of T10I4D5M we systematically modified the parameters L, C and N. In particular, we used L ∈ {1k,10k,100k,1M}, C ∈ {0, 0.5} and N ∈ {1, 10, 100} in the data generation process. The patterns are independent for C = 0, while there is some correlation between them for C = 0.5.

We run the experiments for the values $\tilde{\Delta} = 0.001 + 0.005i$ for $i = 0, 1, \ldots, 9$ and $b = 25$k.[6] For all datasets, we use $\Delta = \lceil \tilde{\Delta} t \rceil$ for the batch and $\Delta = \lceil \tilde{\Delta} s \rceil$ for our streaming algorithm, where s is the sample size. In particular, for $\epsilon = 0.005$ and $\delta = 0.001$ we have $s = 150$k (see Sect. 4.1), corresponding to around 3% of the 5M stream length. The output of our algorithm will be compared to the results obtained by the batch algorithm in terms of precision and recall. That is, denoting by TP the number of strongly closed itemsets found by both algorithms and by FP (resp. FN) that returned only by our algorithm (resp. only by the batch algorithm). Then precision (P) and recall (R) are defined in the standard way, i.e., $P = TP/(TP + FP)$ and $R = TP/(TP + FN)$. The results are reported in Tables 3–5 in terms of precision (P) and recall (R) for the datasets from Table 1 and for the variations of T10I4D5M, together with the number of $\tilde{\Delta}$-closed sets ($|\mathcal{C}_{\Delta,\mathcal{D}_t}|$). We note that for T40I10D5M, the batch algorithm was unable to compute the result for $\tilde{\Delta} = 0.001$ in 24 hours. One can see that the precision and recall values are never below 0.80; in most of the cases they are actually close or equal to 1. The results on the data streams obtained from the benchmark datasets might be favorable for our algorithm due to the repetition of transactions. The two artificial data streams T10I4D5M and T40I10D5M do not have such a bias. Still, we obtained very good results for these data streams as well. Thus the repetition of transactions does not improve the results in favor of our algorithm.

We have carried out experiments on several other artificial data streams generated by the IBM Quest data generator using other parameters selected systematically (except for the size 5M). All results in Tables 4 and 5 are for $N = 1$ (i.e., 1000 items). For larger N the results look similar, but with increasing N the number of strongly closed patterns decreases. The precision and recall values for the synthetic data sets are close to 1, in all considered settings they do not fall below 0.92. Thus our algorithm provides a good approximation of the set of strongly closed itemsets in a transactional data stream.

6 Practical applications

In this section we demonstrate the suitability of strongly closed itemsets and that of our algorithm mining this kind of itemsets in data streams for two practical applications, namely for *concept drift detection* in transactional data streams (Sect. 6.1) and for *computer-aided product configuration* raised by an industrial project (Sect. 6.2).

[6] This value of b is chosen arbitrary. The results in Fig. 12 show that the choice of b is not critical.

Table 4 Number of $\tilde{\Delta}$-closed sets ($|\mathcal{C}_{\Delta,\mathcal{D}_t}|$), precision ($P$) and recall ($R$) after processing 5M transactions of synthetic quest data generated with IBMs Quest data generator

$\tilde{\Delta}$	L = 1k						L = 10k					
	C0			C0.5			C0			C0.5		
	$\lvert\mathcal{C}_{\Delta,\mathcal{D}_t}\rvert$	P	R	$\lvert\mathcal{C}_{\Delta,\mathcal{D}_t}\rvert$	P	R	$\lvert\mathcal{C}_{\Delta,\mathcal{D}_t}\rvert$	P	R	$\lvert\mathcal{C}_{\Delta,\mathcal{D}_t}\rvert$	P	R
0.041	11	1	1	12	1	1	11	1	0.91	11	1	1
0.036	17	1	1	19	1	0.95	20	1	1	24	0.96	1
0.031	33	1	1	34	1	0.97	32	1	1	37	1	0.97
0.026	46	0.98	0.98	54	0.98	1	63	0.95	1	61	0.94	0.98
0.021	75	0.99	1	81	0.98	0.98	101	0.98	0.99	99	0.99	0.99
0.016	132	1	0.98	136	0.97	0.99	164	0.98	0.99	163	0.99	0.98
0.011	226	0.99	0.98	224	0.98	0.97	288	0.99	1	282	0.98	0.99
0.006	392	0.99	0.99	359	0.98	0.99	497	0.99	0.99	471	0.99	0.99
0.001	2618	0.96	0.95	2673	0.95	0.94	2455	0.93	0.94	2590	0.94	0.93

The parameters used in data generation are $T = 10$, $I = 4$, $D = 5M$, and $N = 1$, the remaining parameters are specified in the first two header rows

Table 5 Number of $\tilde{\Delta}$-closed sets ($|\mathcal{C}_{\Delta,\mathcal{D}_t}|$), precision ($P$) and recall ($R$) after processing 5M transactions of synthetic quest data generated with IBMs Quest data generator

$\tilde{\Delta}$	L = 100k						L = 1M1					
	C0			C0.5			C0			C0.5		
	$\lvert\mathcal{C}_{\Delta,\mathcal{D}_t}\rvert$	P	R	$\lvert\mathcal{C}_{\Delta,\mathcal{D}_t}\rvert$	P	R	$\lvert\mathcal{C}_{\Delta,\mathcal{D}_t}\rvert$	P	R	$\lvert\mathcal{C}_{\Delta,\mathcal{D}_t}\rvert$	P	R
0.041	9	0.9	1	8	1	0.88	7	0.88	1	7	1	1
0.036	21	1	0.95	20	1	1	20	1	0.9	20	1	0.95
0.031	34	0.97	0.97	33	0.97	0.97	33	0.97	1	36	1	0.97
0.026	60	0.98	1	59	0.98	1	57	1	0.98	57	0.98	1
0.021	100	1	0.97	104	0.96	0.97	100	0.97	1	100	0.95	0.99
0.016	171	0.98	1	173	0.99	0.98	175	0.98	0.98	175	1	0.99
0.011	311	0.98	0.97	317	0.99	0.98	310	1	1	314	0.98	0.99
0.006	530	0.99	1	535	0.99	0.99	534	0.99	0.99	534	0.99	0.99
0.001	3176	0.93	0.92	3153	0.93	0.93	3260	0.93	0.92	3279	0.92	0.92

The parameters used in data generation are $T = 10$, $I = 4$, $D = 5M$, and $N = 1$, the remaining parameters are specified in the first two header rows

6.1 Concept drift detection

In general, *concept drifts* in data streams are changes in the underlying distribution generating the data observed. Their early detection is an essential step for most practical applications of pattern mining in data streams. As an example, consider the problem of recommending goods to a customer that she is interested in and hence, will purchase with high probability. Computer-aided recommendations often rely on "typical" purchasing patterns extracted from the shopping baskets of other customers. Since these patterns are usually dynamic (i.e., change over time), recommendation systems resorting to typical patterns must work with an up-to-date set of patterns corresponding to the (unknown) current distribution. Motivated by this

and other scenarios, in this section we present an application of strongly closed itemsets to concept drift detection in transactional data streams.

Before going into the details, we would like to stress that the primary goal of this section is *only* to demonstrate the potential of strongly closed itemsets to concept drift detection, and *not* to present a new algorithm specific for concept drift detection. The development of such an algorithm goes far beyond the scope of this work and would require, among others, a mathematically precise specification of a number of components, potentially depending on the data and drift type, such as, for example the sensitivity of the algorithm, the suitable window size, the number of strongly closed sets etc.

Two major classes of concept drifts are distinguished in Li and Jea (2014) for frequent patterns in transactional data streams: *Isolated* and *successive* concept drifts. While isolated concept drifts are single drifts that do not necessarily have any preceding or successive drifts, successive concept drifts are drifts within a drift sequence. The goal of this section is to demonstrate the suitability and effectiveness of strongly closed sets for the first drift type, i.e., for detecting *isolated* concept drifts.[7]

To characterize isolated concepts drifts, the following two dimensions are regarded in Li and Jea (2014):

(i) *pace*, i.e., the time required to completely replace the old distribution by the new one and
(ii) *commonality*, i.e., the overlap of the two distributions.

For (i) we consider both *swift* and *gradual* replacements of distributions. In case of swift drifts, the distribution changes abruptly, i.e., from one transaction to the next. In contrast, gradual drifts have an elongated transition from one distribution to the other. Using these properties, we generate artificial datasets of these two types of replacements as follows: We first create two data streams S_1 and S_2 generated with different distributions. For swift drifts we then simply concatenate S_1 and S_2. For gradual replacements we generate a data stream $S_1 \cdot S \cdot S_2$ by concatenating S_1, S, and S_2, where S consists of ℓ transactions from S_1 and S_2, corresponding to the "graduality" of the drift. In particular, transaction i in S is taken from S_1 at random with probability $1 - i/\ell$ and from S_2 with probability i/ℓ. In this way we simulate a noisy "linear" transition of length ℓ from S_1 to S_2. Clearly, the longer the transition phase the less evident is the exact location of the drift.

For (ii) above we consider *separated* and *intersected* distributions. In case of separated distributions there is no overlap in the distributions. A straightforward way to obtain such distributions is to pick them at random from a pool over pairwise disjoint ground sets. In contrast, intersected distributions are defined over the same ground set and in a way that the individual and the joint probabilities over the ground set are identical for some of the elements and different for the others. We generate both distributions from existing datasets. To generate separated distributions, we simply replaced each item by a new symbol. For intersected distributions, some of the items were removed from the transactions independently and uniformly at random.

Combining (i) and (ii), we thus have four cases for isolated drifts (i.e., swift–separated, swift–intersected, gradual–separated, gradual–intersected). The data streams with concept drifts for our experiments were generated from the datasets in Table 1 by repeatedly drawing transactions from the datasets modifying them as described above. For each data stream, we generated three concept drifts with 2M transactions between any two consecutive drifts; 2M is a sufficiently large length enabling a careful investigation of different features (see below) of our algorithm.

[7] Since our algorithm does not save the concept drifts detected in the past, it is not suited (in its present form) for successive concept drift detection.

To detect the concept drifts in these data streams, we started a new instance of our mining algorithm every 100k transactions, with parameter values $\epsilon = 0.01$ and $\delta = 0.02$. These values give a sample size of around 23k (cf. Sect. 4.1), corresponding to roughly 1% of the 2M transactions between the consecutive drifts. Recall from Sec. 5.2 that we obtained very accurate results for the sample size of 150k. The results in this section obtained for the much smaller sample size of 23k also demonstrate that reliable concept drift detection is possible by means of approximate results. A practical implication of this property is that working with smaller sample sizes allows for faster update times. As indicator for concept drifts, we used the Jaccard distance between the families of strongly closed sets returned by the two consecutive instances of our algorithm (having a delay of 100k); the impact of non-consecutive instances are discussed at the end of this section.

We investigate the effect of different drift characteristics as well as that of different parameter settings of our algorithm. While our aim at considering different drift characteristics is to demonstrate that strongly closed sets can indeed detect a wide range of concept drifts, the analysis of different parameters of our algorithm serves to show that it can detect drifts for various choices of the parameters. That is, the stability of strongly closed itemsets ensures that it is not sensitive to the particular choice of the parameters. In particular, we empirically analyze the effects of drift characteristics for

Drift type the four drift types defined above,
Drift length the length of gradual drifts, and
Drift intersection the probability of overlap for intersected drifts

and those of the algorithm's parameter choices for

Degree of closedness the strength of closedness (i.e., $\tilde{\Delta}$),
Delay the delay after which a new instance of our algorithm is started, and
Buffer the buffer size b of our algorithm.

In order to make our experimental results clearly comparable, we present them in detail only for the Poker-hand dataset. It was selected for this purpose at random out of the six datasets considered in Table 1; for all other five datasets we obtained very similar results for all six characteristics above. Unless otherwise specified, the length of gradual drifts is 250k transactions, the probability for intersected distributions is 0.5, and $b = 25$k. We justify the particular choice of these values by noting that the length of gradual drifts is longer than the sample size, a probability of 0.5 results in a clear contrast between the distributions, and the buffer size was chosen at random close to the sample size.

Drift type Figure 7 presents the results obtained by our algorithm for the four types of isolated drifts. The three drifts are clearly identifiable in all four cases. Notice that for the two swift drifts, the peaks are more spiky than for the two gradual ones. This is due to the reason that in case of swift drifts, the transition from one distribution to the next is more abrupt compared to gradual drifts that spread over more transactions. Comparing the two separated drifts (LHS) with the two intersected ones (RHS), one can observe that the peaks stand more apart for separated drifts. This meets our expectations, as for separated drifts there is (much) less overlap in the data distributions than for intersected ones.

Drift length Figure 8 is concerned with the influence of the drift's length for gradual–separated concept drifts. That is, we are interested in the ability to detect drifts for different times needed to completely replace a new concept with the previous one. We present our results for drift lengths of 1000, 5000, 25,000, and 100,000 transactions. The results

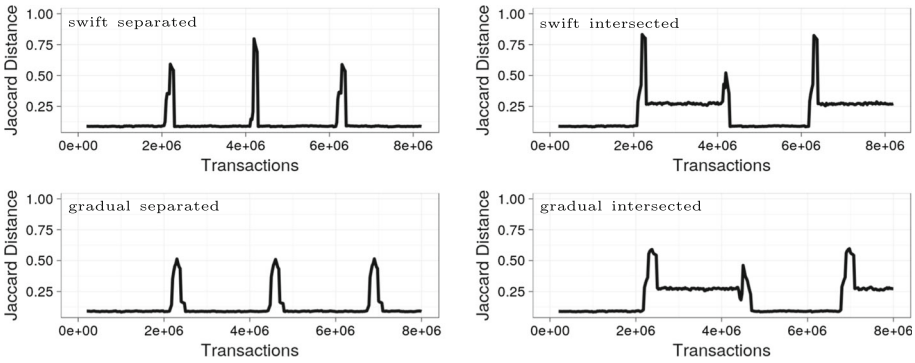

Fig. 7 Concept drift detection results for Poker-hand drift type for swift-separated, swift-intersected, gradual-separated, gradual-intersected at $\tilde{\Delta} = 0.001$

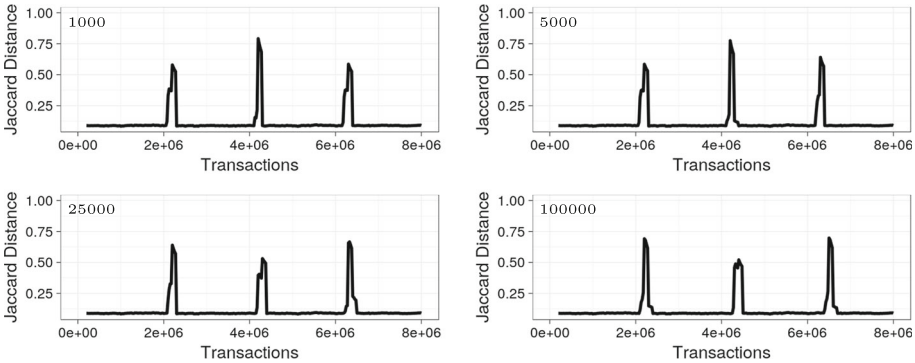

Fig. 8 The influence of the drift length for 1k, 5k, 25k, 100k on concept drift detection. Results for Poker-hand with gradual-separated concept drifts at $\tilde{\Delta} = 0.001$. Note that drifts of length 0 correspond to the swift-separated case

clearly demonstrate that no matter how long the drift's length, as all three drifts are clearly identifiable for all four lengths. In particular, drifts of 1000 and 5000 transactions are clearly shorter, drifts of 25,000 transactions are a bit longer, and drifts of 100,000 transactions are clearly longer than the sample size. The results obtained for intersected drifts are similar. (Swift concept drifts are not presented, as their drift length is always 0.)

Drift intersection In all other experiments, intersected drifts are generated by taking the transactions one-by-one and removing each item from the transaction at hand independently and with probability $p = 0.5$. It is natural to ask how sensitive is our algorithm for other values of p. To answer this question, we generated intersected drifts for $p = 0.1, 0.2, 0.3$, and 0.4. The results are presented in Fig. 9. One can see that the drifts are clearly recognizable for all cases, i.e., even for $p = 0.1$, although there is no peak for this value (in contrast to the three other values). The figure shows a clear correlation between p and the height of the peaks. The results for gradual drifts look

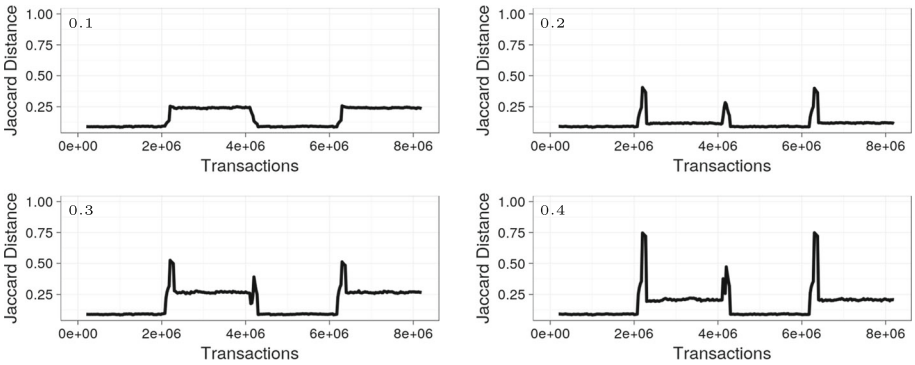

Fig. 9 Concept drift detection results for Poker-hand with swift-intersected concept drifts for probability of intersection for 0.1, 0.2, 0.3, 0.4 and for $\tilde{\Delta} = 0.01$

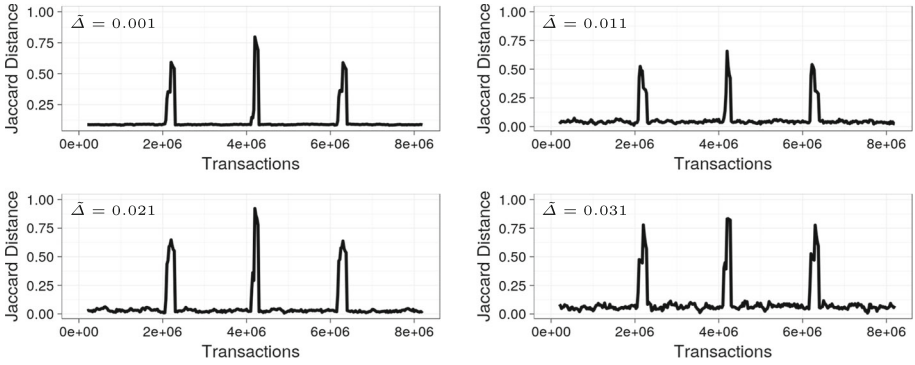

Fig. 10 Concept drift detection results for Poker-hand with swift-separated concept drifts for $\tilde{\Delta} \in \{0.001, 0.011, 0.021, 0.031\}$

very similar with slightly wider peaks (cf. Fig. 7 for the difference between swift and gradual).[8]

Degree of closedness In Fig. 10 we investigate the influence of $\tilde{\Delta}$ ranging from 0.001 to 0.031, corresponding to $\Delta = 23$ and $\Delta = 714$, respectively. The upper limit 0.031 is chosen based on the values in Table 2. In case of Poker-hand for instance, this choice of $\tilde{\Delta}$ results in around 250 (i.e., about 0.5%) strongly closed itemsets out of 46,000 ordinary ones. For all values of $\tilde{\Delta} \in \{0.001, 0.011, 0.021, 0.031\}$ the drifts are clearly visible. While they are smoother and more indicative for lower values of $\tilde{\Delta}$ (i.e., for larger subsets of ordinary closed itemsets), already as few as 250 strongly closed itemsets ($\tilde{\Delta} = 0.031$) suffice to detect the drifts, demonstrating the appropriateness of strongly closed itemsets to concept drift detection.

Delay In Fig. 11 we investigate the effect of the delay, i.e., the number of transactions after which we start a new instance of our mining algorithm. Recall that the Jaccard distance is computed for the output of two consecutively started instances of the algorithm. Clearly, there is some trade-off in choosing the number of transactions between two miners. On the one hand, the more transactions are processed before the next instance of the

[8] Note that a similar experiment is meaningless for separated drifts because they do not share any common items before and after the drift.

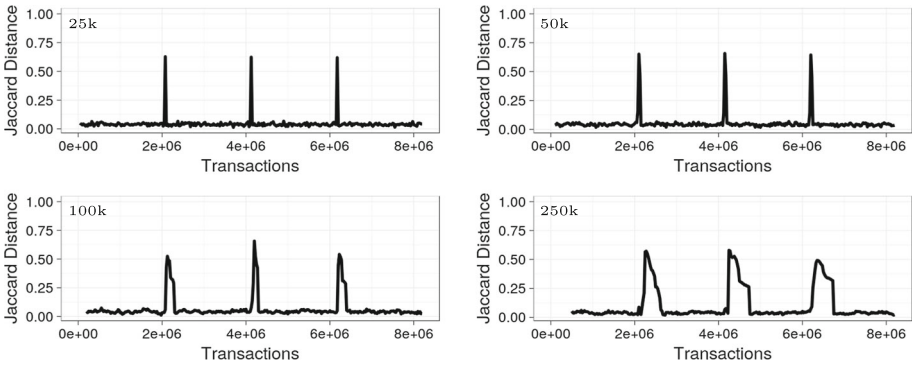

Fig. 11 Influence of the delay between miners for 25k, 50k, 100k, 250k on concept drift detection illustrated for the Poker-hand dataset with swift-separated drift and stongly closed itemsets for $\vec{\Delta} = 0.011$

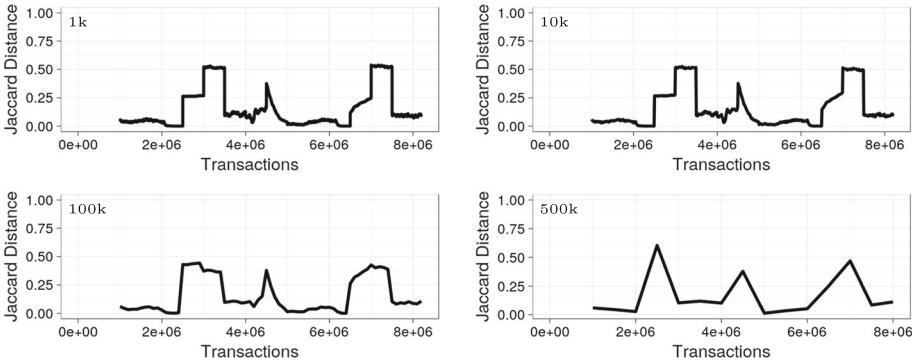

Fig. 12 The effect of the buffer size for 1k, 10k, 100k, 500k on the concept drift detection with strongly closed sets. Results for poker-hand with swift-intersected drifts and $\vec{\Delta} = 0.011$

algorithm is started, the lower the overall runtime. On the other hand, a concept drift can only be located within the interval of transactions between two consecutively started miners. We investigate delays for 25k, 50k, 100k, 250k transactions. Four observations can be made as the delay increases from 25k to 250k. First, for small delays the drifts are detected on the spot, i.e., right when they happen. Second, the peaks, which are very spiky for a delay of 25k transactions become wider for larger delays. Third, the height of the peaks decreases with increasing delay. Forth, as there is more delay between the miners, it takes more transactions after the drift, before it is detected (i.e., the first peak moved from 2.075M for a delay of 25k to 2.25M for a delay of 250k). Still, the drifts are clearly visible in all cases, regardless of the choice of this parameter.

Buffer The effect of the buffer size is shown in Fig. 12. In particular we consider buffers of size 1k, 10k, 100k, 500k. On the one hand, with a larger buffer there are less frequent updates of the family of strongly closed sets, resulting in a better runtime. On the other hand, however, less frequent updates of strongly closed sets reduces the ability to detect drifts close to the time they happen. Our experiments clearly confirm this trade-off. The larger the buffer size, the smoother the plot of the Jaccard distance. (The reason that there is no apparent difference between a buffer of size 1k and 10k is that both are smaller than the sample size.) For buffers of 100k and especially of 500k, the peaks lose there

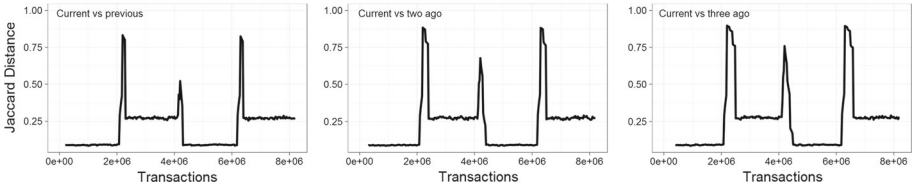

Fig. 13 Concept drift detection results for Poker-hand with swift-intersected concept drifts for varying detection delay at $\tilde{\Delta} = 0.001$

individual shape, but are still clearly visible. In summary, smaller buffers can capture more details of the data stream, while larger buffers result in shorter computation time, as expected.

In all of the experiments so far, we considered Jaccard distances between the outputs of consecutive miners started at equidistant intervals. Even for swift drifts, the drift takes place with high probability during the run of one of the miners. In such cases, the output of the miner might thus not fully reflect the state prior to the drift, but capture already a part of the drift. This is especially true for long intersected drifts. It can thus be favorable to allow for some gap between the outputs of the miners. Figure 13 shows the influence of the gap between two miners. In particular, we compare the current miner with the previously started one (left), with the one started two intervals before (middle), and with the one started three intervals before (right). The three drifts are clearly visible in all settings. With increasing gap between the miners the Jaccard distance reaches higher values in case of drifts, confirming our expectations.

In summary, drift type (Fig. 7) and intersection (Fig. 9) are two drift characteristics having the strongest influence on the Jaccard distance and thus the ability to detect concept drifts by using strongly closed itemsets. In contrast, the particular choice of the parameters of our algorithm seems to show less effect, indicating that our algorithm is not sensitive to them. In particular, for appropriately chosen delays (Fig. 11) between miners, drifts are detected on the spot. The parameters $\tilde{\Delta}$ (Fig. 10) and b (Fig. 12) show the effect that drifts are a little more prominent for lower values. The length (Fig. 8) of the drift seems to have no effect upon the ability to detect drifts. An additional gap between the miners can improve the Jaccard distance (Fig. 13). Having investigated different drift types and the sensitivity of the parameters of our algorithm on various datasets, we finally conclude that strongly closed itemsets are excellent indicators for concept drift detection in data streams.

6.2 Product configuration

As another potential practical application of strongly closed itemsets, in this section we empirically demonstrate their suitability for computer-aided *product configuration*, a problem raised by an industrial project. In particular, we propose an algorithm based on strongly closed itemsets that supports the customer in selecting a set of options (items) from a given pool that together constitute her desired product to be purchased (e.g., an individual composition of a pizza's topping, an individually customized prefabricated house/modular home etc.). Depending on the number of possible options, finding the most appropriate configuration can be a time-consuming and tedious task.

The above kind of configuration problems can be regarded as the following computational problem: Suppose the goal is to identify a product, i.e., an *unknown* transaction $T \subseteq E$

for some finite set E of items. To achieve this goal, the learning algorithm is assumed to have access to a database $\mathcal{D}$ of transactions over E (e.g, pizza toppings, prefabricated houses/modular homes etc. ordered by other customers) and to an *oracle* (i.e., the customer) and it may ask *queries* of the form

$$\text{“Is } Y \subseteq T\text{?”}$$

for some $Y \subseteq E$. In case $Y = T$ (resp. $Y \subsetneq T$) the answer is “EQUAL” (resp. “SUBSET”); otherwise the oracle returns a counterexample $x \in Y \setminus T$. The aim of the learning algorithm is to identify T with as few queries as possible.

Notice that the problem above is in fact *concept learning* with queries. Indeed, just regard E as the instance space and transactions as concepts. For the case that the transaction database $\mathcal{D}$ is not part of the problem setting, exact identification of concepts has systematically been discussed in the pioneering work by Angluin (1987) for various types of queries and concept classes. The query defined in this section can be considered as a combination of equivalence and subset queries (cf. Angluin 1987). The rationale behind considering this type of queries is that most customers have typically some constraint defined in advance for the product to be purchased (e.g., an upper bound on the number of components of the pizza's toping, some fixed budget for the prefab house/modular home etc.) that must be fulfilled by the product. Once the set of items recommended is appropriate for the customer and any further extension would violate the constraint, she might be interested in completing the process (answer “EQUAL”), without considering the remaining options (items) that have not been shown/recommended by the algorithm yet. This is an important requirement especially for such situations where the number of all options or items (i.e., $|E|$) is too large compared to that of the finally selected ones (i.e., $|T|$).

Another difference to the problem settings in Angluin (1987) is that the algorithm has access also to $\mathcal{D}$ containing a set of already purchased configurations. The underlying idea of our approach is that some of the “typical patterns” in $\mathcal{D}$ are likely to be selected also for the unknown target configuration T. It is not difficult to see that for the case when $\mathcal{D}$ is not part of the problem or when the transactions in $\mathcal{D}$ have been generated with an entirely different process as the unknown set T, the number of subset queries required to identify T exactly is $|E| - 1$ if all non-empty subsets of E can be a potential transaction (or concept). In real-world situations both of these assumptions are, however, unnecessarily strong. In fact, as we show empirically using real-world product configuration datasets, the above number can be reduced to $0.5 \cdot |E|$ in average by using strongly closed itemsets.

6.2.1 Algorithm

The algorithm exactly identifying an unknown transaction T over a ground set E of items with queries is given in Algorithm 6. Its input is a database $\mathcal{D}$ of transactions over E and the family $\mathcal{C}_{\Delta,\mathcal{D}}$ of Δ-closed itemsets of $\mathcal{D}$ for some positive integer Δ. Although the algorithm considers $\mathcal{D}$ and $\mathcal{C}_{\Delta,\mathcal{D}}$ as *static* inputs, it can effectively be applied in practice in the data stream setting as well, where $\mathcal{D}$ and $\mathcal{C}_{\Delta,\mathcal{D}}$ are continuously updated as described in Sect. 4. This follows from the properties that $\mathcal{C}_{\Delta,\mathcal{D}}$ contains typically only a few thousands of Δ-closed sets for appropriately chosen Δ and that the time complexity of the algorithm is linear in the combined size of E and $\mathcal{C}_{\Delta,\mathcal{D}}$.

Algorithm 6 starts by initializing the set variable $\mathcal{S}$ with the union of $\mathcal{C}_{\Delta,\mathcal{D}}$ and the family of singleton sets formed by the items in E (Line 1). Some of the singleton sets will be needed for exact identification e.g. in such cases when the unknown transaction T to be identified is

Algorithm 6 EXACT TRANSACTION IDENTIFICATION WITH QUERIES

input: database $\mathcal{D}$ over E and $\mathcal{C}_{\Delta,\mathcal{D}}$ for some $\Delta \in \mathbb{N}$
require: subset query oracle and an unknown set $T \subseteq E$
output: T

1: $\mathcal{S} := \mathcal{C}_{\Delta,\mathcal{D}} \cup \{\{x\} : x \in E\}$
2: $X := \emptyset$
3: $Y := \operatorname{argmax}_{Z \in \mathcal{S}} |Z \setminus X| \cdot |\mathcal{D}[Z]|$
4: call the oracle with query $X \cup Y$
5: **if** ANSWER = "EQUAL" **then return** $T = X \cup Y$
6: **else if** ANSWER = "SUBSET" **then**
7: $\quad X := X \cup Y$
8: $\quad$ remove all sets Z from $\mathcal{S}$ with $Z \subseteq X$
9: **else**
10: $\quad$ let x be the counterexample returned by the oracle
11: $\quad$ remove all sets Z from $\mathcal{S}$ with $x \in Z$
12: **go to** 3

not Δ-closed. In the set variable X we store the set of items of T to be identified that have already been detected by the algorithm. It is initialized by the empty set (Line 2). At this point, the uncertainty as to T is $|E|$ bits. The goal of the learning algorithm is to reduce this amount of uncertainty to zero. To achieve this, on the one hand we prefer queries that reduce the amount of uncertainty with as many bits as possible, i.e., we are interested in selecting a set $Y \in \mathcal{S}$ maximizing $|Y \setminus X|$. On the other hand, however, the larger the cardinality of the query the smaller the chance is that it is a subset of T. Therefore, we need to take into account the absolute frequency (or support count) of the set inquired as well. Since cardinality is at odds with frequency, we control the trade-off between them by the product of the potential information gain with the absolute frequency, and select the set Y from $\mathcal{S}$ maximizing this heuristic (cf. Line 3). As we will see shortly, each set in $\mathcal{S}$ will be queried at most once. We then call the oracle with the union of the already learned subset X of T with this candidate set Y (Line 4) and, depending on its answer, proceed as follows: We stop the algorithm by returning $X \cup Y$ if it is equal to T (Line 5). If Y is a subset of T (Line 6), we add it to X (Line 7) and remove all sets from $\mathcal{S}$ that are contained by X (Line 8), as none of them can further contribute to the reduction of the uncertainty as to T. Note that by definition, the set Y used in the query will also be removed. Finally, if $X \cup Y \not\subseteq T$, the oracle returns a counterexample $x \in Y \setminus T$ (Line 10). As $x \notin T$, we remove all sets in $\mathcal{S}$ that contain x. Note that in this case the amount of uncertainty is reduced by one bit only, in contrast to the case that $X \cup Y \subsetneq T$ (Lines 6–8).

We will compare the performance of our algorithm described above to the following less sophisticated algorithm, called Algorithm BASELINE, obtained from Algorithm 6 by replacing Line 1 with

$$1' : \mathcal{S} := \{\{x\} : x \in E\} \ .$$

That is, this algorithm ignores all Δ-closed sets and uses only singletons in the queries, preferring them by their absolute frequency. The brute-force solution to this problem would be to ask a membership query for all items in some arbitrary order. The difference between this brute-force strategy, referred to as Algorithm NAIVE and Algorithm BASELINE is that Algorithm BASELINE asks the membership queries for the items in the order of their frequencies and can stop the algorithm as soon as the target transaction has been identified. One can

Table 6 Real-world product configuration dataset characteristics

ID	1	2	3	4	5	6	7	8	9	10	11
$\|E\|$	246	239	251	262	334	331	232	237	171	168	154
$\|\mathcal{D}\|$	8341	15,844	19,310	28,239	19,550	50,134	27,078	33,933	9149	17,935	5902
k	52.28	55.29	51.25	43.98	59.95	60.72	48.94	67.14	48.86	47.14	51.62

The three rows correspond to the cardinality of the ground set ($|E|$), number of transactions ($|\mathcal{D}|$), and average transaction size (k)

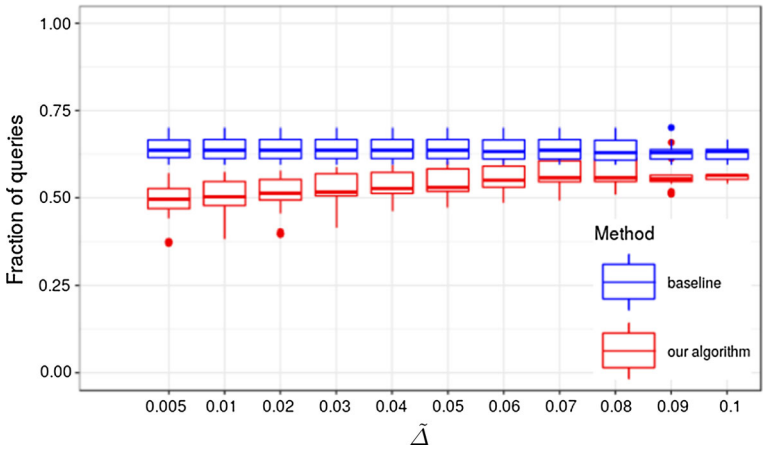

Fig. 14 Product configuration results for varying $\tilde{\Delta}$. Averages over all input datasets

easily see that all three algorithms are correct and require $|E|$ queries in the worst-case.[9] Below we show on real-world datasets that Algorithm 6 requires much less queries than Algorithm BASELINE.

6.2.2 Experimental results

We ran both Algorithms 6 and BASELINE on 11 real-world product configuration datasets from a single real-world product configuration database provided by our industrial partner. (Recall that Algorithm NAIVE would ask a membership query in some ad hoc order for every element of E to identify the unknown target transaction T.) Table 6 contains the cardinality of the ground set ($|E|$), the number of transactions ($|\mathcal{D}|$), and the average transaction size (k) for each of the 11 datasets. For both algorithms we measure the fraction of queries they require compared to Algorithm NAIVE.

Using five-fold cross validation we computed the family of strongly closed itemsets for each of the datasets, used them in Algorithm 6 to identify the transactions in the test set, and calculated the fraction of queries required in comparison to Algorithm NAIVE. Figure 14 shows the average fraction of queries over all datasets required by our algorithm for various values of $\tilde{\Delta}$ and by Algorithm BASELINE. One can see that the number of queries monotonically increases with $\tilde{\Delta}$ in the observed interval, motivating the choice of small values for $\tilde{\Delta}$.

[9] Assuming that transactions are non-empty subsets of E, this worst-case bound can be reduced to $|E| - 1$ by querying finally the set containing the two items left.

This is not surprising, as smaller values of $\tilde{\Delta}$ result in larger families of strongly closed itemsets, allowing for an ultra fine grade of queries. In particular, for $\tilde{\Delta} = 0.005$ our approach requires on average only a fraction of 0.49 of the queries required by Algorithm NAIVE, compared to the fraction of 0.64 needed by Algorithm BASELINE. This results in a saving of 23.4% on average and 37.61% in the best case for Algorithm 6 over Algorithm BASELINE. Note, however, that there is a trade-off between the choice of $\tilde{\Delta}$ and the time of updating the family of strongly closed datasets.

The experimental results of this section clearly demonstrate the potential of strongly closed itemsets on computer-aided product configuration tasks. In the next section we discuss some potential ideas for further improving the query complexity of Algorithm 6. The elaboration of these ideas go beyond the scope of this paper and is left for future work.

7 Concluding remarks

We have presented a general purpose algorithm for mining strongly closed itemsets from transactional data streams under the landmark model. Our algorithm heavily utilizes some of the nice algebraic and algorithmic properties of this kind of itemsets. The speed-up and approximation results presented in Sect. 5 clearly indicate the suitability of our algorithm for mining strongly closed itemsets even from *massive* transactional data streams. The empirical results of the previous section provide also evidence that strongly closed itemsets are of high practical relevance amongst others to concept drift detection in transactional data streams and to computer aided product configuration. These advantageous properties follow from the compactness and stability of strongly closed itemsets.

The speed-up results reported in Sect. 5 can further be improved by utilizing that $|\mathcal{C}_{\Delta,\mathcal{D}_t}|$ is typically (much) smaller than the sample size (s) calculated by Hoeffding's inequality (see (Boley et al. 2009) for a detailed discussion on the size of $\mathcal{C}_{\Delta,\mathcal{D}_t}$). In such cases, the closure $\sigma_{\Delta,\mathcal{D}_{t'}}(C \cup \{e\})$ can be computed from $\mathcal{C}_{\Delta,\mathcal{D}_t}$ without any database access to $\mathcal{D}_{t'}$, even when the closure of $C \cup \{e\}$ has not been calculated for $\mathcal{D}_t$. For example, instead of computing $\sigma_{\Delta,\mathcal{D}_t}(C \cup \{e\})$ in Line 2 of Algorithm 3, we can return $\bigcap\{Y \in \mathcal{C}_{\Delta,\mathcal{D}_t} : C \cup \{e\} \subseteq Y\}$, as $\mathcal{C}_{\Delta,\mathcal{D}_t}$ is a closure system.

The impressive experimental results of Sect. 6.1 strongly motivate the design of an algorithm *specific* to concept drift detection that is based on mining and monitoring the changes in strongly closed itemsets. Besides the landmark model considered in this work, the problem of mining strongly closed itemsets under the *sliding window* model would be an interesting related problem. The solution of this problem requires, however, an entirely different algorithmic approach. Another interesting question is to develop an algorithm optimally adjusting the strength of closedness (i.e., $\tilde{\Delta}$) during concept drift detection.

Similarly to the remark above, the algorithm proposed in Sect. 6.2 for the product configuration task can further be improved by designing an algorithm *specific* to this problem. Potential improvements towards this research direction include the utilization of partial orders on the ground set, extensions of the problem by Boolean and cost constraints, and a more sophisticated treatment of the items used in the queries, once the subset queries have been exhausted and the algorithm automatically switches into asking subset queries formed by singletons. (We recall that the only difference to ordinary membership queries lies in the answer EQUAL). For the queries in this state of the algorithm, it would be essential to utilize some appropriate linear order on the set of uncertain items remained (i.e., which have so far been neither included nor excluded). A very natural candidate for such an order could be

defined by the conditional probabilities of items with respect to strongly closed sets. When the family of strongly closed sets is of small cardinality, which is typically the case, these conditional probabilities could completely be calculated/updated and stored in a feasible way.

The question discussed above for product configuration raises a general problem. Instead of calculating the closure system for some particular value of Δ, it would be interesting to maintain a closure system formed by Δ-closed itemsets for *different* values of Δ. Such a closure system should fulfill the condition that $\Delta_1 \geq \Delta_2$ whenever a Δ_1-closed itemset precedes a Δ_2-closed itemset in it. In the context of product configuartion, this would allow for working gradually with more and more strongly closed itemsets, resulting in a further reduction in the number of membership queries asked by our algorithm in its second phase.

Acknowledgements The authors thank Claus-Peter Buszello for useful discussions on the computer-aided product configuration task. Part of this work has been funded by the Ministry of Education and Research of Germany (BMBF) under project ML2R (grant number 01/S18038C).

References

Angluin, D. (1987). Queries and concept learning. *Machine Learning*, *4*(2), 319–342.

Bai, S. P., & Kumar, G. K. R. (2016). A survey on closed frequent itemset mining on data streams. In *Proceedings of the 2nd International Conference on Contemporary Computing and Informatics (IC3I)* (pp. 542–547).

Boley, M., Horváth, T., Poigné, A., & Wrobel, S. (2010). Listing closed sets of strongly accessible set systems with applications to data mining. *Theoretical Computer Science*, *411*(3), 691–700.

Boley, M., Horváth, T., & Wrobel, S. (2009). Efficient discovery of interesting patterns based on strong closedness. *Statistical Analysis and Data Mining*, *2*(5–6), 346–360.

Boros, E., Gurvich, V., Khachiyan, L., & Makino, K. (2003). On maximal frequent and minimal infrequent sets in binary matrices. *Annals of Mathematics and Artificial Intelligence*, *39*(3), 211–221.

Chi, Y., Wang, H., Yu, P. S., Muntz, R. R. (2004). Moment: maintaining closed frequent itemsets over a stream sliding window. In *Proceedings of 4th International Conference on Data Mining (ICDM)* (pp. 59–66).

Dua, D., & Graff, C. (2019). *UCI Machine Learning Repository*. Irvine, CA: University of California, School of Information and Computer Science. http://archive.ics.uci.edu/ml/index.php.

Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM Computing Surveys*, *46*(4), 44:1–44:37.

Ganter, B., & Reuter, K. (1991). Finding all closed sets: A general approach. *Order*, *8*(3), 283–290.

Gély, A. (2005). *A generic algorithm for generating closed sets of a binary relation* (pp. 223–234). Berlin: Springer.

Gupta, A., Bhatnagar, V., & Kumar, N. (2010). Mining closed itemsets in data stream using formal concept analysis. In *Proceedings of 12th International Conference on Data Warehousing and Knowledge Discovery*, DaWaK (pp. 285–296).

Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, *58*(301), 13–30.

Iwanuma, K., Yamamoto, Y., & Fukuda, S. (2016). An on-line approximation algorithm for mining frequent closed itemsets based on incremental intersection. In *Proceedings of the 19th International Conference on Extending Database Technology*, (pp. 704–705).

Jiang, N., & Gruenwald, L. (2006). CFI-Stream: Mining closed frequent itemsets in data streams. In *Proceedings of the 12th ACM SIGKDD*, (pp. 592–597).

Kifer, D., Ben-David, S., & Gehrke, J. (2004). Detecting change in data streams. In *Proceedings of the 13th International Conference on Very Large Data Bases (VLDB)*, (pp. 180–191).

Knuth, D. E. (1997). *The art of computer programming. Vol. 2: Seminumerical algorithms*. Reading: Addison-Wesley.

Li, C.-W., & Jea, K.-F. (2014). An approach of support approximation to discover frequent patterns from concept-drifting data streams based on concept learning. *Knowledge and Information Systems*, *40*(3), 639–671.

Liu, X., Guan, J., & Hu, P. (2009). Mining frequent closed itemsets from a landmark window over online data streams. *Computers & Mathematics with Applications*, *57*(6), 927–936.

Loglisci, C., Ceci, M., Impedovo, A., & Malerba, D. (2018). Mining microscopic and macroscopic changes in network data streams. *Knowledge-Based Systems*, *161*, 294–312.

Manku, G. S., & Motwani, R. (2002) Approximate frequency counts over data streams. In *Proceedings of the 28th International Conference on Very Large Data Bases*, (VLDB), (pp. 346–357). VLDB Endowment.

Pasquier, N., Bastide, Y., Taouil, R., & Lakhal, L. (1999). Efficient mining of association rules using closed itemset lattices. *Information Systems*, *24*(1), 25–46.

Serfling, R. J. (1974). Probability inequalities for the sum in sampling without replacement. *Annals Statistics*, *2*(1), 39–48.

Trabold, D., & Horváth, T. (2017). Mining strongly closed itemsets from data streams. In A. Yamamoto, T. Kida, T. Uno, and T. Kuboyama (Eds.), *Discovery science*. DS 2017. Lecture Notes in Computer Science (Vol. 10558, pp. 251–266). Cham: Springer.

Tsai, P. S. M. (2009). Mining frequent itemsets in data streams using the weighted sliding window model. *Expert Systems with Applications*, *36*(9), 11617–11625.

van Leeuwen, M., Siebes, A. (2008). StreamKrimp: Detecting Change in Data Streams. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases 2008*, (ECML PKDD), (pp. 672–687).

Vitter, J. S. (1985). Random sampling with a reservoir. *ACM Transactions on Mathematical Software*, *11*(1), 37–57.

Yen, S. J., Wu, C. W., Lee, Y. S., Tseng, V. S., Hsieh, C. H. (2011). A fast algorithm for mining frequent closed itemsets over stream sliding window. In *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, (pp. 996–1002)