CHAPTER 2

REVIEW OF RELATED RESEARCH

Frequent pattern mining is a core data mining operation and has been studied extensively in the last decade. Recently mining frequent patterns from high speed data streams has gained vast interest in the research area. In this chapter, the state-of-art techniques, to mine frequent patterns from static datasets and also dynamic data streams, are reviewed. Techniques using different representations of frequent patterns and different models of data streams are also discussed.

2.1 FREQUENT PATTERN MINING

Frequent pattern mining has been a focused theme in data mining research for over a decade. Abundant literature has been dedicated to this research and tremendous progress has been made, ranging from efficient and scalable algorithms for frequent itemset mining in transaction databases to numerous research frontiers, such as sequential pattern mining, structured pattern mining, correlation mining, associative classification, and frequent pattern-based clustering, as well as their broad applications. Frequent pattern mining research has substantially broadened the scope of data analysis and has deep impact on data mining methodologies and applications [51]. Frequent itemsets play an essential role in many data mining tasks [94] that try to find interesting patterns from databases, such as association rules, correlations, classifiers, clusters and many more of which the mining of association rules is one of the most popular problems[28].

Association is the discovery of togetherness or connection of objects [99]. Association rules are used to express this togetherness or connection between objects. The original motivation for searching association rules came from the need to analyze so called supermarket transaction data, that is, to examine customer behavior in terms of the purchased products [10]. In such analysis association rules describe how often items are purchased together. For example a retail store may discover that whenever people buy soft drinks they also buy snacks items like potato chips, samosa, and many more. The owners may decide to advertise combo packs consisting of soft drinks, potato chips and some other items based on such rules. This
will definitely improve the sales margin of the retail store. Such rules can be useful for decisions concerning product pricing, promotions, store layout and many others also. Within the past decade, hundreds of research papers have been published presenting new algorithms or improvements on existing algorithms to solve these mining problems more efficiently.

Most association rule mining algorithms employ a support-confidence framework [17]. Often, many interesting rules can be found using low support thresholds. Although minimum support and confidence thresholds help weed out or exclude the exploration of a good number of uninteresting rules, many rules so generated are still not interesting to the users. Unfortunately, this is especially true when mining at low support thresholds or mining for long patterns. This has been one of the major bottlenecks for successful application of association rule mining. A correlation measure can be used to augment the support-confidence framework for association rules to tackle this weakness.

Correlation is often used as a preliminary technique to discover relationships between variables. More precisely, the correlation is a measure of the linear relationship between two variables. Pearson’s correlation coefficient is defined as:

\[
\text{corr}(x, y) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\left[ \sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2 \right]^{1/2}}
\]

This leads to correlation rules of the form: \( A \rightarrow B \) \{support, confidence, correlation\}

That is, a correlation rule is measured not only by its support and confidence but also by the correlation between itemsets A and B. There are many different correlation measures from which to choose. The most important are lift and \( \chi^2 \) correlation measures.

Frequent pattern mining was initially proposed by Agrawal et al. in 1993 [2] for market basket analysis in the form of association rule mining. It analyses customer buying habits by identifying associations between the different items that customers purchase. For instance, if customers are buying milk, how likely are they going to also buy cereal (and what kind of cereal) on the same trip to the
supermarket? Such information can lead to increased sales by helping retailers do selective marketing and arrange their shelf space. Agrawal’s initial proposal was followed by many follow-up algorithms which are extensions of the Apriori algorithm for a variety of applications, and variety of data types. With over a decade of substantial and fruitful research, it is still a flourishing field with new algorithms written for frequent itemset discovery and lot more has to be done before this area can be treated as a cornerstone in data mining applications.

2.1.1 Apriori principle, apriori algorithm and its extensions

There are usually a large number of distinct single items in a typical transaction database, and their combinations may form a very huge number of itemsets. So it is challenging to develop scalable methods for mining frequent itemsets in a large transaction database. Agrawal and Srikant in 1994 proposed the Apriori algorithm [3] that employed downward closure property, called Apriori property to overcome this challenge. Apriori property states that a k-itemset is frequent only if all of its sub-itemsets are frequent. This implies that frequent itemsets can be mined by initially finding all the unique items in the database and finding their frequency of occurrence. These items constitute the 1-itemsets and are found by scanning the database once. The infrequent 1-itemsets, items whose frequency of occurrence is less than a user-specified threshold, are pruned before proceeding further. Since the Apriori property confirms the fact that the infrequent 1-itemsets cannot lead to frequent 2-itemsets they can be safely pruned. Then using the frequent 1-itemsets, candidate 2-itemsets are generated, and the database is scanned to obtain the frequency of these candidate 2-itemsets. If the frequency of the candidates is greater than the specified threshold, then they are added to the set of frequent 2-itemsets. This process iterates until no more frequent k-itemsets can be generated for some k. This is the essence of the Apriori algorithm proposed by Agrawal and Srikant in 1994 [3] and its alternative algorithm proposed by Mannila et al. in 1994 [64]. Since the Apriori algorithm was proposed, there have been extensive studies on the improvements or extensions of Apriori. The various improvements suggested were hashing technique, partitioning technique, sampling technique, dynamic itemset counting and incremental mining.
Hashing technique was employed in the algorithm, DHP (Direct Hashing and Pruning) proposed by Park et al. in 1995 [69]. This algorithm proved to be more efficient for large itemset generation. It employs a hash method for generating candidates in the earlier iterations, especially for 2-itemset candidates and also employs pruning techniques to reduce the database size. In 1995 Savsere et al. [80] proposed a partitioning technique which helped to reduce the Input/Output (I/O) overhead and the Central Processing Unit (CPU) overhead for most cases. This was possible since the partitioning algorithm employed, divided the large database logically into a number of non-overlapping partitions and found potentially large itemsets in each partition in one scan. In a subsequent scan the actual support of these itemsets was calculated and the actual large itemsets were identified. This technique helped in reducing the size of the candidate set to be checked.

In 1996 Toivonen developed a sampling technique [86] to reduce the I/O overhead. The author proposed to pick a random sample from the database, find the association rules from them and validate them with the rest of the database. Thus even though only a sample of the data is used, exact association rules are produced and not approximations based on the sample. Also in the same year an incremental updating technique was suggested by Cheung et al. [17] for maintenance of association rules discovered from dynamic databases using Fast Update (FUP) algorithm.

In 1997 Dynamic Itemset Counting (DIC) algorithm [10] was introduced by Brin et al. to find large itemsets with less number of scans over the data than classical algorithms. The authors used the idea of item reordering in their algorithm, DIC to improve the low-level efficiency and also presented a new way of generating implication rules.

### 2.1.2 Mining frequent itemsets without candidate generation

In many cases, the Apriori algorithm significantly reduces the size of candidate sets using the Apriori principle. However, this algorithm can suffer from two non-trivial costs [33]. They include the huge number of candidates generated when the support threshold is low and the repeated scanning of databases required to find the frequency count of the candidates generated.
In the year 2000 Han et al. devised a Frequent Pattern Growth method (FP-growth) [33] that mines the complete set of frequent itemsets without candidate generation. FP-growth works in a divide-and-conquer way. The initial scan of the database derives a list of frequent items in which items are ordered by frequency in descending order. According to the frequency-descending list, the database is compressed into a frequent pattern tree (FP-tree), which retains the itemset association information [1]. The FP-tree is mined by starting from an initial suffix pattern. Each frequent 1-itemset forms an initial suffix pattern. Next for each such pattern, its conditional pattern base is constructed. It is a “sub database”, which consists of the set of prefix paths in the FP-tree co-occurring with the suffix pattern. Finally conditional FP-trees are constructed from the conditional pattern bases and mining is performed recursively on each such tree. The pattern growth is achieved by the concatenation of the suffix pattern with the frequent patterns generated from a conditional FP-tree. The FP-growth algorithm transforms the problem of finding long frequent patterns to searching for shorter ones recursively and then concatenating the suffix. It uses the least frequent items as a suffix, offering good selectivity. Performance studies demonstrate that the method substantially reduces search time.

Agarwal et al. proposed an extension to the FP-growth approach, the depth-first generation of frequent itemsets [1] in the year 2001. In this Tree Projection algorithm, the frequent itemsets are represented as nodes of a lexicographic tree. The lexicographic tree is built in a depth-first manner. The root of the tree represents the projection of the entire transaction database. Once this projection is complete, finding sub-trees rooted at a given node and pruning off unnecessary branches may be performed as independent tasks with a substantially reduced transaction set. In this approach, CPU time is reduced at the increased cost of disk I/O [1].

Also in 2001, Pei et al. introduced Hyper-structure Mine (H-Mine) [71] that explores a hyper-structure, mines frequent patterns and introduces a new mining technology, the space-preserving mining technology. It exhibits limited and precisely predictable space overhead. It can scale up to very large databases by using database partitioning and it can switch to building alternative FP-trees in case of dense datasets.
In the year 2002, Liu et al. proposed OpportuneProject algorithm [59] to mine the complete set of frequent itemsets by projecting databases, which is efficient on both sparse and dense databases at all levels of support threshold, and scalable to very large databases. The algorithm builds frequent itemset tree in a depth-first manner but shifts to breadth-first manner if necessary when building the top portion of the tree.

Grahne and Zhu proposed an array-based technique [30] that greatly reduces the need to traverse FP-trees, thus obtaining significantly improved performance of FP-tree based algorithms in 2003. The technique works especially well for sparse datasets. They proposed algorithms that use the FP-tree data structure in combination with an array technique efficiently, and incorporate various optimization techniques.

2.1.3 Mining frequent itemsets using vertical data format

Both the Apriori and FP-growth methods mine frequent patterns from a set of transactions in horizontal data format, \{TID: itemset\}, where TID is a transaction identifier and itemset is the set of items bought in the transaction TID. Alternatively, mining can also be performed with data presented in vertical data format, \{item: TID _set\}.

In 2000, Mohammed Zaki proposed Equivalence CLAss Transformation (ECLAT) algorithm [99] by exploring the vertical data format. The first scan of the database builds the TID _set of each single item. Starting with a single item \(k = 1\), the frequent \((k+1)\)-itemsets are generated from previous frequent \(k\)-itemsets according to the Apriori property, with a depth-first computation order similar to FP-growth [33]. Intersecting the TID _sets of the frequent \(k\)-itemsets is done to compute the TID _sets of the corresponding \((k+1)\)-itemsets. This process repeats, until no frequent itemsets or no candidate itemsets can be found.

The TID _set of each \(k\)-itemset carries the complete information required for counting support. This makes it possible to find the support of \((k+1)\)-itemsets (for \(k \geq 1\)) without any further scanning of the database. This is an added advantage gained in addition to that gained by using the Apriori property.
In 1995, Holsheimer et al. published another related work which mines the frequent itemsets with the vertical data format [36]. The authors used a decomposed storage structure, in which each transaction has a unique transaction identifier, and the database is stored as a set of items (columns), where for each item the TIDs of the transactions that contain this item are enumerated. This work demonstrated that, though impressive results have been achieved for some data mining problems using highly specialized and clever data structures, solving data mining problems using the general purpose database management systems is also highly possible and provides an avenue for huge improvement in data mining techniques [54].

2.1.4 Mining closed and maximal frequent itemsets

A major challenge in mining frequent patterns from a large dataset is the fact that such mining often generates a huge number of patterns satisfying the minimum support threshold, especially when it is set low [55]. This is because if a pattern is frequent, each of its sub patterns is frequent as well. A large pattern will contain an exponential number of smaller, frequent sub-patterns. To overcome this problem, closed frequent pattern mining [101] and maximal frequent pattern mining was proposed [28]. A pattern ‘\(\alpha\)’ is a closed frequent pattern in a dataset \(D\) if ‘\(\alpha\)’ is frequent in \(D\) and there exists no proper super-pattern \(\beta\) such that \(\beta\) has the same support as ‘\(\alpha\)’ in \(D\). A pattern ‘\(\alpha\)’ is a maximal frequent pattern (or max-pattern) in set \(D\) if ‘\(\alpha\)’ is frequent, and there exists no super-pattern \(\beta\) such that \(\alpha \subset \beta\) and \(\beta\) is frequent in \(D\). For the same minimum support threshold, the set of closed frequent patterns contains the complete information about all the frequent patterns that can be derived from it [89]. But the set of max-patterns usually does not contain the complete support information but it is more compact than closed frequent patterns.

In 1999 Pasquier et al. first proposed the mining of closed frequent itemsets (CFIs). The authors proposed an Apriori-based Close (A-Close) algorithm [70] for mining CFIs using a closed itemset lattice framework. A-Close is efficient for mining dense and/or correlated data. It also performs relatively well with weakly correlated data. CLOSET (CLOsed itemSET) [72] is another efficient CFI mining algorithm that uses a compressed FP-tree structure that enables mining CFIs without the costly candidate generation of Apriori-based algorithms like A-Close.
Closed Association Rule Mining (CHARM) [100] is another CFI mining algorithm that explores both the itemset space and the transaction space simultaneously which helps to skip many levels and quickly identify the CFIs. It makes fewer scans over the database than the longest CFI found and it scales linearly with number of transactions and number of closed itemsets found. CLOSET+ [93], is another CFI mining algorithm. CLOSET algorithm grows patterns by projection of conditional databases in a bottom-up manner but CLOSET+ employs bottom-up projection for dense datasets and top-down projection for sparse datasets which helps to achieve better performance for different kinds of datasets. It scales better than CLOSET and CHARM algorithms. CLOSET+ also performs well when compared to CLOSET and CHARM when number of distinct items increases. FPClose (Frequent Pattern based Closed Itemset mining) algorithm [30] also mines CFIs but unlike CLOSET+, which constructs one global tree, FPClose uses multiple, conditional CFI-trees for checking closedness of itemsets. This helps FPClose outperform CLOSET+.

The main challenge in closed / maximal frequent pattern mining is to check whether a pattern is closed /maximal. There are two strategies to approach this issue: (1) to keep track of the TID list of a pattern and index the pattern by hashing its TID values. This method is used by CHARM which maintains a compact TID list called a diffset; and (2) to maintain the discovered patterns in a pattern-tree similar to FP-tree. This method is exploited by CLOSET+, and FPClose. Mining closed itemsets provides an interesting and important alternative to mining frequent itemsets since it inherits the same analytical power but generates a much smaller set of results. Better scalability and interpretability is achieved with closed itemset mining. Memory consumption is limited.

DCI-CLOSED (Direct Count & Intersect – Closed itemset) algorithm [60] employs divide-and-conquer technique and bitwise representation of the vertical database. The pruning technique used in this algorithm makes it possible for only part of the closed patterns to be in main memory rather than the entire set. It also allows independent mining of the partitions which pave the way for parallel mining. The main memory consumed by this algorithm is constant unlike FPClose and CLOSET+ since these algorithms require the entire set of closed itemsets extracted so far to be in main memory at any instant unlike DCI-CLOSED algorithm [60].
Also the DCI-CLOSED can mine any dataset as long as its vertical representation fits into the main memory, irrespective of the number of Closed Itemsets (CIs) extracted.

Index-FCI (Index based Frequent Closed Itemset mining) algorithm [35] also mines Closed Frequent Itemsets (CFIs) from databases. The authors use three techniques to improve the performance of the algorithm. The database is compressed and BitTable is constructed in increasing support order. An index array corresponding to the database is maintained. Great Frequent Itemset (GFI) is found with the help of this index array. All these techniques reduce the time for CI checking. The CIs are maintained as a hash table with their support as the hash value. Hash pruning is used to remove the frequent but non-closed itemsets in this algorithm. Index-FCI runs faster than FPClose with reduced support thresholds since FPClose needs more time to construct conditional FP-trees.

DBV-Miner (Dynamic Bit Vectors) algorithm [90] constructs a DBV-tree and mines CFIs. It employs DBVs to store the vertical representation of each item. This helps in reducing the memory required when compared with algorithms that use a fixed-size bit vector to represent the items. Also a technique for computing the intersections between DBVs at greater speed is used. A Lookup table is employed to reduce the time needed for computing number of bits after finding the intersections. These techniques make DBV-Miner algorithm more efficient in terms of mining time and memory consumption when compared with CHARM algorithm.

Mining max-patterns was first studied by Bayardo in 1998 who proposed Max-Miner [9], an Apriori-based, level-wise, breadth-first search method to find max-itemset, by performing superset frequency pruning and subset infrequency pruning for search space reduction. The performance improvement shown by Max Miner algorithm over other Apriori-like algorithms is very high when FIs are long and moderate when they are short. Another efficient method MAFIA (MAximal Frequent Itemset Algorithm) [11], proposed by Burdick et al. in 2005, uses vertical bitmaps to compress the TID list, thus improving the counting efficiency. The parent equivalence pruning and dynamic reordering techniques helped to reduce the search space. The relative compression/projection of the vertical bitmaps dramatically reduced the cost of counting supports and increased the vertical scalability of MAFIA.
GenMax (Generate Maximal Frequent Itemset) algorithm [28] is a backtracking algorithm for mining Maximal Frequent Itemsets (MFIs). It uses progressive focusing to perform maximality testing and diffset propagation to perform fast frequency computation. MAFIA returns a superset of MFI but GenMax returns the exact MFI.

2.2 MINING DATA STREAMS

More and more applications such as traffic modeling, military activities sensing and tracking, online data processing and many more, generate huge flows of data every day. Efficient knowledge discovery from such data streams is an emerging active research area in data mining with ever growing applications [44]. Different from data in traditional static databases, data streams typically arrive continuously at high speed [15] in huge volumes and varying distribution. This raises new issues that need to be considered when developing mining techniques for stream data. Owing to the unique features of data stream, traditional data mining techniques which require multiple scans of the entire datasets cannot be applied directly to mine stream data, since stream data usually allows only one scan and demands fast response time [8].

A data stream is an ordered sequence of items that arrives in timely order. Different from data in traditional static databases, data streams are continuous, unbounded, usually come with high speed and have a data distribution that often changes with time. As the number of applications on mining data streams grows rapidly, there is an increasing need to perform various mining operations on stream data. For most data stream applications, there are needs for mining frequent patterns and association rules from data streams. Some key applications in various areas are listed below [31].

Performance monitoring

Monitoring network traffic and performance helps in finding the total network utilization that helps in detecting congestions and deciding whether network capacity should be increased or congestion control techniques should be applied. They can also help in detecting abnormality in traffic patterns that play an important role in intrusion detection. Packets flowing across the Internet can be monitored to develop a general understanding of the Internet traffic patterns which is essential for
efficient network routing, caching, pre-fetching, network upgrades and many more. Monitoring the packet streams also help in predicting the frequency of packets in a data stream that has numerous applications in Internet routers and gateways like implementing fairness policy which helps in preventing some users monopolizing the network.

**Transaction monitoring**

Transactions in retail stores are monitored regularly. Transactions originating from Automated Teller Machines (ATM) are also monitored to detect fraudulent transactions and theft of ATM cards. Log records of financial markets are monitored and mined periodically to identify buying and selling patterns that may help to predict the behavior of stock markets. Telecommunication calling records are also monitored continuously to help in crime detection. Web server log files are monitored and reports are generated from the web log streams that may help in recognizing visiting patterns of users. Association rule mining can also be applied to monitor manufacturing flows in automated industries to predict failure and also generate alarms when malfunctioning is detected.

**Sensor network mining**

Advancements in developing low power sensors have lead to the deployment of large-scale sensor networks all over the world. They produce large scale, continuous data streams that have to be monitored. Mining interesting patterns from such streams plays a vital role in many applications in today’s world. Some such applications are habitat-monitoring, object tracking, environment monitoring, military, and disaster management.

**2.2.1 Data Stream Classifications**

Data streams can be classified into offline streams and online streams. Offline streams are characterized by regular bulk arrivals. Data arrives in bursts and not as continuous streams [78]. Among the applications discussed in the earlier section, generating reports based on web log streams can be treated as mining offline data streams because most of reports are made based on log data in a certain period of time. The log records are permanently stored and may be processed at a later
convenient time. Other offline stream examples include queries on updates to warehouses or backup devices. Queries on these streams are allowed to be processed offline during slack time or after multiple queries have been accumulated that can be processed simultaneously with relatively few accesses.

Online streams are characterized by real-time updated data that come one by one in time. Among the above examples, predicting frequency estimation of Internet packet streams is an application of mining online data streams because Internet packet streams is a real-time packet-by-packet process. Other online data streams are stock tickers, network measurements and sensor data [54]. They have to be processed online and must keep up with the rapid speed of online queries. Once they arrive they are processed and then immediately discarded. In addition, unlike offline data streams, bulk data processing is not possible for online stream data.

Research on data streams started around 2000 when several data stream management systems were initiated including the Brandeis Aurora project, the Cornell Cougar project, and the Stanford Stream project. Data stream management systems were developed to solve new challenges in applications such as network traffic monitoring, transaction data management, Web click streams monitoring, sensor networks, and many more.

Because association rule mining plays an important role in these data stream applications, along with the development of data stream management systems, developing association rule mining algorithms for data streams has also become an important research topic.

2.2.2 General Issues in Data Stream Association Rule Mining

Algorithms for association rule mining usually consist of two steps. The first step is to discover frequent datasets. In this step, all frequent datasets that meet the support threshold are discovered. The second step is to derive association rules. In this step, based on the frequent datasets discovered in the first step, the association rules that meet the confidence criterion are derived [3]. Since deriving association rules can be solved efficiently in a straightforward manner once the frequent itemsets are available, most of researches focus mainly on the first step, i.e., how to efficiently discover all frequent itemsets in data streams. Therefore the focus of this research
will also be on frequent itemset mining in data streams rather than association rule mining.

Frequent itemset mining in general is already a challenging problem. For example, due to combinatorial explosion, there may be huge number of frequent itemsets, and a main challenge is how to efficiently enumerate, discover, and store frequent itemsets. Data streams, because of their unique features, have further posed many new challenges to frequent dataset mining [83].

Recently, the data generation rates in many data sources have become faster than ever before. This rapid generation of continuous streams of information has challenged the storage, computation and communication capabilities in computing systems. Thus the storage, querying and mining of data streams have become highly computationally challenging tasks. This has made data stream mining a stimulating field of study that has raised numerous challenges and research issues to be addressed by the database and data mining communities.

2.3 LITERATURE SURVEY

In this section, a number of representative state-of-the-art algorithms on mining frequent itemsets, frequent maximal itemsets, or frequent closed itemsets over data streams are surveyed. The stream mining algorithms are organized into three categories based on the window model that they adopt: the landmark window, the damped window or the sliding window. Each window model is then classified as time-based or count-based. According to the number of transactions that are updated each time, the algorithms are further categorized into update per transaction or update per batch algorithms. The mining algorithms are classified based on results obtained into two categories: exact or approximate. The approximate algorithms can be further classified into algorithms based on the false-positive approach and the false-negative approach. False-positive approach returns a set of itemsets that includes all frequent itemsets but also some infrequent itemsets, while false-negative approach returns a set of itemsets that does not include any infrequent itemsets but misses some of the frequent itemsets [98]. The different issues resulting from the different window models and the nature of the algorithms are discussed in this
section. The underlying principles of the algorithms are explained and their merits and limitations are brought out.

2.3.1 Landmark window based algorithms

In landmark window model shown in Fig. 2.1, frequent pattern mining is performed based on the values between a specific time-stamp, called landmark, and the present.

![Fig. 2.1 Landmark Window Model](image)

Manku and Motwani [63] proposed the Sticky sampling and lossy counting algorithms for computing an approximate set of FIs over the entire history of a stream in 2002 which uses BTS. BTS refers to the three modules namely Buffer, Trie and SetGen used in these algorithms. Sticky sampling algorithm takes samples and constructs a synopsis over a data stream of singleton items. A data structure S is constructed containing the entries of the form (e, f), where ‘f’ estimates the frequency of item e in the data stream. The algorithm is called sticky sampling because as it scans the stream it attracts the elements already in S like a magnet. It guarantees to produce all items whose frequency exceeds ‘f’.

Lossy counting is a deterministic algorithm that computes frequency over a data stream of transactions having only one item [63]. The performance of this algorithm is better than that of Sticky Sampling algorithm. An estimation mechanism is used in the Lossy Counting algorithm which is a classic algorithm for mining frequent datasets in data streams and is based on the well-known Apriori property: if any length ‘k’ pattern is not frequent in the database, its length (k+1) super-patterns can never be frequent, to harmonize such a conflict. The Lossy Counting algorithm
mines Frequent Itemsets (FIs) in data streams, when a maximum allowable error ‘ε’ as well as a minimum support ‘σ’ is given and the number of transactions seen so far in the data stream, ‘N’ is known. The information about the previous mining results up to the latest block operation is maintained in the data structure called lattice [48].

A distinguishing feature of the Lossy Counting algorithm is that it outputs a set of datasets that have the following guarantees:

- All FIs are output. There are no false-negatives
- No itemset whose actual frequency is less than \((\sigma - \varepsilon)N\) is output
- The difference between the computed frequency of an itemset and its actual frequency never exceeds \(\varepsilon N\)

However, using a relaxed minimum support threshold, ‘ε’ to control the quality of the approximation of the mining result leads to a dilemma. The smaller the value of ‘ε’, the more accurate is the approximation but the greater is the number of sub-FIs generated, which requires both more memory space and more CPU processing power. However, if ‘ε’ approaches ‘σ’, more false-positive answers will be included in the result, since all sub frequent-FIs whose computed frequency is at least \((\sigma - \varepsilon)/N \approx 0\) are output while the computed frequency of the sub-FIs can be less than their actual frequency by as much as \(\sigma^*N\). This problem also exists in other mining algorithms that use a relaxed minimum support threshold to control the accuracy of the mining result.

StreamMining algorithm [42] is an in-core frequent dataset mining algorithm based on BTS algorithm. The algorithm uses a hybrid approach. It reduces the memory requirements by reducing the number of frequent 2-itemsets. This algorithm uses the reduced set of frequent 2-itemsets and the apriori property to further reduce the number of i-itemsets, for i>2 and establishes a bound on the false positives. To store and manipulate the candidate frequent itemsets during any stage of the algorithm, a lattice is maintained. It is a single pass algorithm for frequent itemset mining in a streaming environment. This algorithm takes as input two input parameters, ‘θ’, the minimum support level and ‘ε’, the factor that determines the accuracy of the algorithm. Given the desired support level ‘θ’, this one pass algorithm reports all datasets occurring with frequency level ‘θ’ and does not include
any dataset occurring with frequency level less than \((1-\varepsilon)|D|\), where \(|D|\) is the number of transactions seen so far in the stream. In the process, the memory requirements increase by a factor proportional to \(1/\varepsilon\). This algorithm uses a data structure, referred to as TreeHash. This data structure implements a prefix tree using a hash table. It has the compactness of a prefix tree and allows easy deletions like a hash table.

This is a one pass algorithm for streaming environment, which has deterministic bounds on the accuracy. Particularly, it is the first such algorithm which does not require any out-of-core memory structure and is very memory efficient in practice. One exception is that datasets with very large average length of an itemset are not processed efficiently. Extra knowledge about maximal frequent itemsets is required in such cases. The memory requirements of this algorithm are significantly lower than those of FP-tree. However, time-sensitive queries have not been considered in this algorithm [5].

DSM-FI (Data Stream Mining for Frequent Itemsets) algorithm proposed in 2008 [56] reads a basic window of transactions from the buffer in main memory and sorts them in lexicographical order. It constructs and maintains an in-memory prefix-tree based summary data structure called SFI-Forest (Summary Frequent Itemset Forest). It prunes the infrequent information from the current SFI-Forest. It finds the Frequent Itemsets (FIs) from the SFI-Forest when it is needed. The frequency error guarantees provided by this algorithm is same as that of BTS algorithm. An efficient FI search mechanism called ToDoFIS (Top Down Frequent Dataset Search) is used to identify FIs from the SFI-Forest. DSM-FI has three major features: single streaming data scan for counting frequency of itemsets, extended prefix-tree based compact pattern representation, and top-down FI discovery scheme.

It is efficient on both dense and rare datasets. An SFI-forest is a more compact data structure than a prefix tree. The number of CFIIs approaches that of FIs when the stream becomes large. Moreover, the compactness of the data structure is paid for by the price of a higher computational cost, since more tree traversals are needed to gather the frequency information of the itemsets.
hMiner (Hash Based Miner) Algorithm proposed in 2009 [91] uses landmark window model to mine frequent itemsets over data streams using an online-processing mode. This algorithm uses a hash based approach combined with the principles of Lossy Counting [63]. The algorithm continuously hashes the subsets contained in the newly arrived transaction. This helps to construct a synopsis of the data stream called hSynopsis, consisting of a hash table and frequent itemsets only. The transactions in the data stream need not be stored.

Use of online-processing mode helps to avoid the need for specifying the maximum length of frequent itemsets to be found in advance. Also the use of probabilistic approach guarantees that the error rate is no more than $\epsilon*N$. The extra processing needed by hCount to retrieve the frequent itemsets and the extra memory needed by Lossy Count in maintaining the non-frequent itemsets as a varying sized list is avoided in this algorithm. The advantages of hCount and Lossy Count are combined here to overcome the disadvantages of both these algorithms. But the maintaining time and memory of hMiner is affected by ‘$\epsilon$’ and ‘$\sigma$’, the error rate and support threshold. Some out of date frequent itemsets may also be maintained by hMiner.

Anamika Gupta et al. [31] have proposed an algorithm, CLICI, (Concept Lattice based Incremental Closed Itemset) for mining all recent closed itemsets in a landmark window model of the online data stream in the year 2010. It fades out the obsolete information of old transactions using a decay function and later using an offline component the decayed information is pruned. The algorithm consists of an online component, which processes the transactions arriving in the stream without candidate generation and updates the synopsis appropriately. The offline component is invoked on demand to mine all frequent closed itemsets and also remove the decayed information. The closed itemsets are stored in a synopsis structure called CILattice (Closed Itemset Lattice).

The algorithm ensures the generation of only non-redundant association rules with the help of the lattice based synopsis, CILattice. Also the data stream needs to be scanned only once to maintain the synopsis. The user can explore and experiment by specifying the support threshold dynamically since the lattice stores all the closed itemsets regardless of their support. But when the pruning threshold
decreases the size of the synopsis increases which leads to an increase in the rate of growth of per-transaction-processing-time.

### 2.3.2 Damped window based algorithms

Damped-window model shown in Fig. 2.2 gives more weightage to recent transactions and less weightage to old transactions. In damped windows, the most recent windows are more important than the previous ones. In other words, older transactions contribute less towards the dataset frequencies [46]. Tilted-time window helps to model the fact that people are interested in recent changes at a fine granularity but long term changes at a coarser level.

![Fig. 2.2 Damped Window Model](image)

**Fig. 2.2 Damped Window Model**

EstDec (estimating Recent Frequent Itemsets based on Decay Rate) algorithm [12] is an approximate damped window based algorithm for mining data streams proposed in the year 2003. The estDec method examines each transaction in a data stream one by one without any candidate generation and keeps track of the occurrence count of an itemset in the transactions generated so far by a prefix tree structure.

EstDec maintains a lattice for recording the potentially frequent itemsets and their counts, and updates the lattice for each new transaction correspondingly. By recording additional information for each itemset p, the time-point of the most recent transaction that contains p, the algorithm only needs to update the counts for the itemsets which are the subsets of newly arrived transactions. It will reduce their counts using the constant factor d and then increase them by one.
The use of a decay rate diminishes the effect of the old and obsolete information of a data stream on the mining result. However, estimating the frequency of an itemset from the frequency of its subsets can produce a large error and the error may propagate all the way from the 2-subsets to the n-supersets, while the upper bound is too loose. Thus, it is difficult to formulate an error bound on the computed frequency of the resulting itemsets and a large number of false-positive results will be returned, since the computed frequency of an itemset may be much larger than its actual frequency. Moreover, the update for each incoming transaction (instead of a batch) may not be able to handle high-speed streams.

FP-Stream algorithm [26] is an algorithm using FP-tree based model for mining frequent patterns from data streams proposed in the year 2004. It uses damped window model also called tilted-time window model. This model helps to mine time-sensitive patterns with approximate support guarantee. Each pattern is maintained at multiple time granularities in tilted-time windows. One FP-tree is used in this algorithm, where at each node, the frequency for each tilted-window representing time at different granularities is maintained. The update of the FP-tree is done in batches not for each new transaction arriving in the stream.

The time and space complexities of the algorithm stabilize as the stream progresses. This provides evidence that the algorithm can handle long data streams efficiently. It is found that the space requirements of this algorithm easily fit into the main memory.

INSTANT (maxImal frequenT So-far ITemset mAIrTainer) is also a single-phase algorithm for finding MFIs [62] over a data stream proposed in the year 2007. It stores all the current maximal itemsets for a data stream INSTANT keeps $F$ as a set of maximal frequent itemsets at any time. Given a continuous data stream, a shedding condition and a user-specified minimum support count $\delta$, INSTANT consists of the following four steps:

1. Generate a sorted itemset list from the current transaction of the data stream
2. Form new frequent itemsets and update the set of frequent itemsets
3. Modify information about infrequent elements and their supports based on $\delta$
4. Execute a shedding plan to maintain search efficiency and memory usage
Steps 1 and 2 must be performed online when new itemsets arrive. Step 3 can be conducted as a background process. Step 4 can take place periodically or on need based. Given a predefined minimum support count ‘δ’ (δ ≥1), the algorithm uses δ -1 distinct arrays. Each array U[i], 1 <= i <= δ -1 stores those maximal infrequent itemsets whose support counts are the same as ‘i’. In addition, the array F maintains all Maximal Frequent Itemsets (MFI) regardless of the support of each.

A new MFI can be displayed as soon as it becomes a frequent itemset. However, the algorithm has some serious drawbacks. First of all, the algorithm employs a number of arrays to maintain all MFI and maximal infrequent itemsets so that the required size of memory space is very large [65]. In addition, since there is no efficient superset/subset checking mechanism for a newly identified MFI against the itemsets of each array, the number of comparisons among itemsets is increased and the required amount of memory space is enlarged rapidly when the average length of transactions becomes longer. Although correctness and completeness of the resulting set of MFI can be guaranteed for every incoming transaction, the algorithm sacrifices both processing time and memory usage and so the algorithm is not suitable for an online data stream.

Li & Gong proposed Top-k-FCI (Top-K Frequent Closed Itemsets) algorithm [48] in the year 2011 to mine the top-k closed itemsets from data streams using damped data stream sliding window model. The algorithm used an extended lattice structure called CIL (Closed Itemset Lattice). CIL consisted of four substructures: a list of bit-vectors of items (BV-List), a set of closed itemset tree (CI-Tree), a list of supports of closed itemsets(S-List), and a hash table of closed itemsets (HTC). The algorithm utilizes a damped factor, ‘ε’, with values in the range (0, 1] to create a dampening effect on the old transactions in the window. The window is divided into basic windows for each of which a CIL is constructed.

Top-k-FCI is an efficient algorithm for mining the top-k-frequent itemsets [48]. The candidate generation technique used in this algorithm helps in making the algorithm more time and space efficient. It also helps to avoid the problem of data bumps. But when the number of windows in the dataset increases building and maintaining the CIL takes longer processing time [51]. Space consumption is also high when insertion and deletion of transactions is performed.
2.3.3 Sliding window based algorithms

The inspiration behind sliding window model shown in Fig. 2.3 is that the user is more concerned with the analysis of most recent data streams [52]. Thus the detailed analysis is done over the most recent data items and summarized versions of the old ones. This idea has been adopted in many techniques. In the sliding window model, knowledge discovery is performed over a fixed number of recently generated data items which are the target of data mining.

![Fig. 2.3 Sliding Window Model](image)

Some of the benchmark algorithms that employ sliding window model are described in the following sections.

2.3.3.1 Sliding Window based algorithms for FI Mining

estWin (estimating Recent Frequent Itemsets in a sliding Window) algorithm [13] proposed in the year 2003 is used for finding recently frequent itemsets adaptively over an online transactional data stream when a minimum support $S_{min} \in (0, 1)$, a significant support $S_{sig} \in (0, S_{min})$ and the size of a sliding window ‘$w$’ are given. Significant itemsets, are maintained in main memory by a lexicographic tree structure. The effect of information in an old transaction that goes out of the range of the sliding window is eliminated by decreasing the occurrence count of each itemset that appeared in the transaction. The estWIN algorithm can be employed to find the recent change of embedded knowledge in a data stream. The interesting recent range of a data stream is defined by the size of a sliding window.

The total number of itemsets monitored in main memory is minimized by employing delayed-insertion and pruning techniques. It is also useful in identifying
the recent changes in a data stream. The sliding window size can be changed to suit the user’s requirement regarding the range of recent change. The method provides flexible trade-off between processing time and mining accuracy. But the main drawback is that the old disregarded information is not available if in future there is a need to analyze them. Since no summary structure about the old transactions is maintained.

Li & Lee proposed the MFI-TransSW and MFI-TimeSW algorithms [52] in the year 2009. MFI-TransSW algorithm (Mining Frequent Itemsets within a Transaction-sensitive Sliding Window), is used to mine the set of frequent itemsets online and incrementally within a sliding window by using an effective bit-sequence representation of items. Based on the same MFI-TransSW framework, an extended single-pass algorithm, called MFI-TimeSW algorithm (Mining Frequent Itemsets within a Time-sensitive Sliding Window), is used to mine the set of frequent itemsets over data streams within a time-sensitive sliding window [57].

In MFI-TransSW algorithm, for each item ‘X’ in the current transaction sensitive sliding window (TransSW), a bit sequence with ‘w’ bits, denoted as Bit(X), and is constructed. If an item X is in the i\textsuperscript{th} transaction of current TransSW, the i\textsuperscript{th} bit of Bit(X) is set to be 1; otherwise, it is set to be 0. This process is called bit-sequence transform. MFI-TransSW algorithm consists of three phases: window initialization, window sliding and frequent itemsets generation.

The major differences between MFI-TransSW and MFI-TimeSW are the unit of data processing, the bit-sequence transformation of a time unit, the number of sliding transactions, and the dynamic frequent threshold of itemsets [65]. In the MFI-TransSW algorithm, the constant value s*w is the frequent threshold of datasets, where ‘s’ is the user specified minimum support threshold in the range of [0, 1] and ‘w’ is the size of sliding window. However, in the MFI-TimeSW algorithm, the value of frequent threshold is the product of minimum support threshold, ‘s’ and the number of transactions in the window. Here the second term, the number of transaction in the window, is a dynamic value.

MFI-TransSW is an efficient single-pass algorithm for mining the set of frequent itemsets over data streams with a transaction-sensitive sliding window.
effective bit-sequence representation of items is developed to enhance the performance of MFI-TransSW. MFI-TransSW and MFI-TimeSW not only attain highly accurate mining results, but also run significantly faster and consume less memory than do existing algorithms, such as SWF (Sliding Window Filtering) algorithm used for mining frequent itemsets from data streams within a sliding window.

The main drawback is that they maintain all frequent itemsets which is quite a large number. Instead they can go for storing only the closed frequent itemsets which will reduce the space requirement. Also these algorithms employ the candidate generation concept of Apriori algorithm which can be improved upon by employing methods which do not go for candidate generation.

Sliding Window based Combinatorial Approximation (SWCA) algorithm [50] is an approximation algorithm for mining frequent itemsets over sliding windows in a data stream proposed in the year 2011. Combinatorial Approximation technique is used in this algorithm that helps to approximate the counts of itemsets based on some already gathered information called base summary. The size of base-summary is set to two and so all subsets of the first-two orders contained in the data stream and their occurrences are recorded in the base summary. This algorithm also processes the sliding window in a segment oriented manner. The sliding window is divided into ‘m’ segments, where a segment is a sequence of fixed number of transactions and sliding takes place by removing the oldest segment and inserting the most recent segment.

The authors showed that approximate support counts of itemsets can be calculated from the counts of their subsets instead of scanning the entire transactions. The dynamic change in minimum support value helps the user to achieve more preferable results unlike the fixed minimum support value used in many algorithms which is impractical for many real-life applications [63]. The segment oriented processing of the sliding window is beneficial when compared to the transaction sensitive sliding process since the amount of processing involved during updates is low in the former than the latter sliding. This is because in transaction-sensitive sliding the update of transactions is excessively frequent. The main drawback is that as the itemset whose count has to be approximated grows in length, the error of approximation increases progressively.
List based Data Stream Mining (LDS) algorithm [21] is a sliding window based algorithm for mining frequent patterns over data streams proposed by Deypir and Sadreddini in the year 2012. This algorithm provided a dynamic layout of sliding window. A set of three simple lists, tidlist, difflist and bitlist are used for storing the occurrence information of items existing within the window. The list representation of an item is not fixed but it changes according to the current support value of that item. For each item in the window depending on its frequency of occurrence, the most efficient list is chosen. If the frequency of an item is less than 1/L, where L is the number of transactions seen in the data stream, tidlist representation is used. If the frequency is more than 1-1/L then difflist representation is used. If the frequency is more than 1/L and less than 1-1/L then bitList representation is used. A window adjustment technique helped in controlling memory usage when concept changes occurred in the data stream. When the user requested for frequent patterns, depending on the current content of the window a suitable mining approach was chosen from among the three versions of Eclat algorithm: Eclat, dEclat and bEclat.

The algorithm worked in sliding window model but can easily adopted for landmark window model by neglecting the deletion procedure which was used to remove the oldest transaction in the sliding window. The memory usage of the algorithm is also almost constant for varying window sizes. This is because the algorithm used simple list structures to create a vertical representation of the transactions and also shifted from one representation to another depending on the current frequency count of the items. But the list conversion time is significant in this algorithm if concept change in the data stream is high. But for data streams with low and moderate concept changes, this algorithm was found to be effective and efficient.

Li et al. proposed Dynamic Base Combinatorial Approximation (DBCA) algorithm [51] in the year 2012. It is a load-controllable mining algorithm for discovering frequent patterns in a transactional data stream using a sliding window model that uses segment-based updates. To handle peak load in the data streams two overload-handling mechanisms are used in this algorithm. The first mechanism aims at accelerating the data handling process and the second aims at reducing the quantity of unprocessed data. This helps the algorithm to work normally during high-data-load-periods also. Also from the stream of data, the first few orders of itemsets are extracted and kept as the global information base. The size of the base information is
the number of orders of itemsets being extracted and saved. This base information is used for approximating the counts of itemsets whose length is greater than the base size.

This algorithm dynamically determines the size of base information for different data stream sources which makes count approximation more accurate. The dynamic extraction and maintenance of base information helps in accelerating the processing of data for overload handling. High data processing rate is achieved by DBCA because it maintains only small-scale base information and approximates the count of other itemsets. The workload of DBCA is independent of minimum support value due to its fixed base threshold.

But the overload-handling mechanisms of DBCA helps handle peak loads only as emergency measures. These mechanisms are effective only when the peak load occurs occasionally and not always. If the data stream is always overloaded then DBCA is insufficient and hence more powerful algorithms are needed.

Variable Size Sliding Window (VSW) algorithm [22] proposed by Deypir et al. in the year 2012 is used for frequent itemset mining in data streams. The window size is determined dynamically based on the rate of concept change in the data stream. The window expands when the concept of the data stream does not change and shrinks when there is a concept change.

The dynamic window size used in this algorithm helps overcome two issues in the sliding window model. The constant window size of a sliding window proves to be a problem since the old transactions are removed irrespective of whether a concept change has occurred or not. Another problem is determining the correct window size. This needs expert knowledge about the behavior of the data stream [22]. Both these problems are avoided when the window size is determined dynamically based on the current behavior exhibited by the data stream. The reduced window size when concept change occurs helps in reducing the memory consumed and also better reflects the changes in the frequent patterns. The algorithm is also able to detect abrupt as well as gradual changes in the concept of the data stream.
Sohrabi and Barforosuh proposed the Systolic Array Based Mining (SABMA) Algorithm [81] in the year 2013. It is an efficient and scalable parallel mining algorithm that mines frequent itemsets in parallel using divide and conquer technique. In this algorithm, a bit matrix structure is used to compress the data and also to model the pattern mining problem as a systolic array problem. Systolic arrays help in implementing the parallel architecture. In addition to the bit matrix structure this algorithm maintains two additional lists, colSum and rowSum. colSum is maintained for each column (item) of the bit matrix and rowSum is maintained for each row (transaction) of the bit matrix. The dataset can be pruned by removing all columns whose colSum value is less than the minimum support threshold.

This algorithm is used for mining frequent itemsets (MiFIs) from very large and high dimensional databases efficiently.

### 2.3.3.2 Sliding window based algorithms for CFI Mining

Chi et al. proposed a new algorithm, MOMENT, (Maintaining Closed Frequent Itemsets by Incremental Updates) [18] to mine and store all closed frequent- itemsets in the year 2006. It uses a sliding window which holds the most recent samples in a data-stream. The recent samples are stored efficiently using memory-structure called the Closed-Enumeration-Trees (CET). The CET is constructed using a depth first process which maintains every itemset in lexicographical order and if the itemset is found frequent then it is added to the tree as a node. Non-frequent itemsets are also stored as they may become frequent in the future. This approach distinguishes between 4 types of nodes, namely, Infrequent gateway node (infrequent nodes whose parents and/ siblings are frequent), Unpromising gateway node (infrequent node who has a superset having same support as itself), Intermediate node (frequent node who has a superset having same support as itself), and Closed node (node which has no children with greater or equal support as itself).

During the sliding window timeframe, whenever a transaction arrives, MOMENT classifies the nodes present in the transaction into one of the 4 categories mentioned previously. Whenever the user requests for mined results the CET (which
observes only closed nodes and boundary nodes) is traversed to mine the closed frequent itemsets.

Unlike prefix trees, MOMENT’s CET maintains only closed itemsets and nodes that mark the boundary between closed itemsets and the rest of the itemsets. This reduces the number of nodes to a large extent when compared to prefix trees. But still the CET data structure operations such as tree creation and maintenance is very time-consuming (since it scans the tree in a depth-first manner), the storage consumed and computational overheads incurred by MOMENT are very high. Thus, it can be surmised that, although algorithms based on exact methods guarantee better accuracy they suffer greatly in terms of computation time and memory requirements. Therefore, several approximation based methods were initiated to provide satisfactory performance by keeping computation costs under control.

NewMoment algorithm [49] proposed by Li et al. in the year 2009 is used to mine closed frequent itemsets in data streams with a transaction sensitive sliding window. A bit-sequence representation of items is used in this algorithm to reduce the time and memory needed for the sliding of windows observed in MOMENT algorithm. A modified summary structure called NewCET is used in this algorithm which is an extended prefix tree structure.

The NewCET structure used in the NewMoment algorithm is more compact than the CET structure used in MOMENT algorithm. The loading time of the first window is also lesser in NewMoment because generating candidates and counting their supports with bit-sequences is faster than with independent sliding windows used in MomentFP, and FPTree algorithms.

Hua-Fu Li proposed an interactive single-pass algorithm, TKC-DS (Top_K frequent Closed itemsets of Data Streams), in the year 2009 [47]. It is used for mining top-K closed itemsets from data streams within a transaction-sensitive sliding window efficiently. This algorithm uses CIL (Closed Itemset Lattice), a data structure to maintain essential information of frequent closed itemsets (FCIs) generated so far. CIL consists of three substructures, BV-List, CI-Tree and S-list for storing the bit vectors of items, closed itemset tree and supports of closed itemsets
respectively. It also employs a Hash Table of Closed itemsets (HTC) for checking whether an itemset is closed or not.

TKC-DS runs faster than Moment-K and NewMoment-K which are the modified MOMENT and NewMoment algorithms that generate not all closed frequent itemsets but only the top-K closed frequent itemsets. The space consumption of TKC-DS is better than Moment-K and NewMoment-K. This is because Moment-K needs to maintain additional information about infrequent, unpromising and intermediate nodes unlike TKC-DS and more number of candidates are generated by NewMoment-T than TKC-DS. Also TKC-DS runs faster than Moment-K and NewMoment-K.

Pauray Tsai proposed the FCI_max (Frequent Closed Itemset) algorithm [88] in the year 2010 for mining top-k frequent closed itemsets of length not more than max_l. Deciding on the minimum support required for finding frequent itemsets is not easy. The value has to be tuned several times before an appropriate value is chosen that helps to generate useful association rules. A number of association rules are generated as a result of mining frequent itemsets above a specified minimum support and many of these rules may be redundant. Both these drawbacks are rectified in FCI_max algorithm as minimum support need not be set and also many redundant association rules may be avoided when the top-K frequent closed itemsets are used to form the association rules.

The TFP (Top-K FCI using FP-Tree) algorithm proposed by the authors in 2005 [92] mined top-k frequent closed itemsets of length not less than min_l. This may result in loss of information about closed itemsets with high support but short length. The method of using maximum length in FCI_max algorithm helps to avoid such information loss.

Tang et al. proposed an algorithm, EMAFCI (Efficient Mining Algorithm for Frequent Closed Itemsets) over data streams [85] in the year 2011. The algorithm is based on the sliding window model, and uses a Bit Vector Table (BVTable) where the transactions and itemsets are recorded by the column vector and row vector respectively. In EMAFCI algorithm, Pair-test operation on binary
numbers in the table is used to detect the Closed Frequent Itemsets (CFIs). The BVTable is updated for each sliding window.

The average time for processing a sliding window with EMAFCI algorithm is less when compared with MOMENT algorithm especially when support is small. The total memory consumed is also less. This is mainly because the search space is reduced to a large extent in EMAFCI when compared with MOMENT and also EMAFCI uses a space efficient BVTable to store the transactions and itemsets whereas MOMENT uses a CET structure that stores four categories of itemsets: infrequent, unpromising, intermediate and closed itemsets which takes up more space than BVTable and also requires more processing time to build and maintain.

AFPCFI-DS (An improved FP tree based algorithm for CFI Mining over Data Streams) algorithm uses sliding window model and is developed by Dai and Chen in 2012 [19]. Whenever a transaction is added or deleted the FP-tree and the GCT-tree are updated independently. AFPCFI-DS builds the Global Closed frequent itemset Tree (GCT) at a faster rate than FPCFI-DS. But the time for building the first window is higher in AFPCFI-DS algorithm than FPCFI-DS (FP tree based algorithm for CFI Mining over Data Streams) [7] algorithm. The performance and space requirement of AFPCFI-DS algorithm is better than MOMENT algorithm. MOMENT algorithm requires huge amount of memory and processing time for building its Closed Enumeration Tree (CET).

In the year 2013, Fatemeh Noria et al. [67] introduced an effective and efficient TMoment (Transaction-Moment) algorithm, for CFI mining over data streams. It operates in the sliding window model and maintains the transactions along with the CIs. This algorithm used a novel data structure, TCET (Transaction translate CET), for storing transactions of the window and corresponding CFIs. Moreover, the support of a new CFI was efficiently computed and an old pattern was removed from the monitoring set when it was no longer a CFI. Extensive experiments on both real and synthetic data streams showed that the algorithm was superior to previously devised algorithms in terms of runtime and memory usage.

MOMENT algorithm has a main drawback that the number of CET nodes is very high when compared to the number of closed frequent itemsets. In
NewMoment algorithm only the closed frequent itemsets and single items are stored. But for some data streams with low or moderate density, the performance of NewMoment is not good since it uses bit strings to represent every item and this becomes costly and consumes large amount of memory. TMoment algorithm overcomes both these drawbacks since it maintains only closed itemsets in TCET and it does not use bit strings to represent items but uses a transaction list.

2.3.3.3 Sliding window based algorithms for MFI Mining

Maximal-Frequent-ItemSets Mining (Max-FISM) algorithm [25] proposed by Farzanyar et al. in the year 2012 is used to mine the recent maximal frequent itemsets from a high speed data stream of transactions within a sliding window. A prefix-tree based summary data structure called Max-Set is maintained by this algorithm to maintain the maximal itemsets. The algorithm utilizes two lists list for maintaining maximal frequent itemsets and minimal infrequent itemsets that are identified by traversing the Max-Set.

This algorithm does not create any false negative errors because the window, the associated summary structure Max-Set, and the two lists are kept refreshed between user requests. Memory consumption is also less since the summary structure is very compact. Max-FISM consumes less memory than MOMENT as the number of nodes maintained is less in Max-FISM since only the maximal itemsets are maintained unlike MOMENT in which four categories of nodes are maintained: closed frequent nodes, infrequent nodes, intermediate nodes and unpromising nodes. Max-FISM also consumes less memory than MFI-TransSW because MFI-TransSW algorithm requires space for bit-sequence representation of all items in the window and also for other parameters like name of the items and frequency count of each item. The space requirement for MFI-TransSW is significantly high when the window size is large as the bit-sequence size is same as the window size.

2.3.4 Max-frequency window model

Generating Global Approximate Closed Frequent Itemset on Max Frequency Window model (GGACFI-MFW) algorithm proposed by Guo et al. in the year 2011 [32] is used to find out the CFIs efficiently over data streams. In this
algorithm, a Max-Frequency Pattern tree (MFP-Tree) is used to maintain a synopsis of the entire stream. Also this algorithm uses a selective update mechanism to traverse the MFP-Tree with high efficiency and make the updating process more efficient. An approximation ratio is used to control the size of the mining results. A cache tree is established to maintain buffer transactions.

The three well known data stream processing models - landmark window model, sliding window model, and the damped window model require the experience of experts for some important parameters. Landmark window model requires the parameter, landmark (a specific time in the past) to be set carefully. The window length in a Sliding window model is a key parameter set by experts. The damping factor, ε has to be set carefully by experts in damped window model. This algorithm uses a relatively new model called Max-Frequency Window model where no such parameters need to be set by experts. Also it maintains the global frequent itemsets over the entire stream.

### 2.3.5 Weighted Sliding Window model

In the year 2009, Pauray proposed the Weighted Sliding Window (WSW) algorithm [87]. The WSW model used in this algorithm is shown in Fig. 2.4 and it is a flexible window model suitable for continuous query processing in data streams. The time interval for periodical queries is set as the window size. It allows the user to specify the number of windows, the size of a window and the weight of each window. This helps the users because it enables them to set a higher weight for more significant portions which in turn leads to results tailored to the user requirements.

![Fig. 2.4 Weighted Sliding Window Model](image)
WSW algorithm efficiently discovers all frequent itemsets from data streams and uses a weighted sliding window model. An improvement algorithm, WSW-Imp helps to reduce the time further by reducing the time of deciding whether a candidate itemset is frequent or not.

The problems faced by data stream mining algorithms in general and frequent pattern mining algorithms in specific have been studied in detail. Research works involving different stream models have been surveyed and it is identified that there is still scope for improving the efficiency of frequent pattern mining algorithms in data streams.