CHAPTER II

RELATED WORK

In this chapter, previous work done on the lines of the three problems taken up for study are discussed. The first two problems – viz. the task of mining maximal k-frequent intervals (Chapter IV) and the task of mining closed k-frequent intervals (Chapter V) – are both related to the notion of k-frequent intervals. The notion of k-frequent intervals has been derived from that of frequent itemsets. Frequent itemsets were introduced by Agrawal et al. in [AIS93] to discover association rules in a transactional database. A transactional database is a database of transactions, where each transaction is a set of items. A subset s of the complete set of items in a transactional database is said to be an itemset. An itemset containing k items is said to be a k-itemset. A non-empty itemset is said to be a frequent itemset if it is contained in at least a *minsup* fraction of transactions in a transactional database, where *minsup* is a user-specified threshold. [AIS93] addresses the problem of discovering useful associations among the items in a transactional database. To solve this problem, at first, all frequent itemsets are determined. Since the total number of possible itemsets is exponential with respect to the number of items in the complete set of items in a transactional database, the task of mining frequent itemsets is in itself a very challenging task. In [AS94], Agrawal and Srikant propose an algorithm called Apriori, which is a level-wise breadth-first algorithm for mining frequent itemsets. The algorithm intelligently trims the exponential search space by eliminating as early as possible, itemsets that are guaranteed to be infrequent and by

generating candidate itemsets that are likely to be frequent. The Apriori algorithm mines frequent itemsets by first scanning the database to find the frequent 1-itemsets, then using the frequent 1-itemsets to generate candidate frequent 2-itemsets, next, checking the database to obtain the frequent 2-itemsets and so on. This process iterates until no more frequent k-itemsets can be generated for some k. The time and space complexity of the Apriori algorithm has been studied by Dunkel and Soparkar [DS99]. Ever since the Apriori algorithm was introduced, there have been several attempts to design more efficient algorithms for mining frequent itemsets. Many of these algorithms share a couple of underlying key features with Apriori in that they also generate candidate itemsets and require multiple database passes to obtain the frequent itemsets. These algorithms however attempt to reduce the total number of database scans that are required to mine all the frequent itemsets and accordingly, they differ from Apriori in the manner in which they handle the candidate itemsets. The DIC (Dynamic Itemset Counting) algorithm proposed by Brin et al. [BMUT97] adds candidate itemsets at different points during a database scan, unlike the Apriori algorithm, which determines candidate itemsets only immediately before a complete database pass. The DIC algorithm usually requires fewer database passes than the Apriori algorithm. The Partition algorithm developed by Savasere et al. [SON95] finds candidate itemsets by partitioning the database into n non-overlapping partitions. The algorithm determines all the frequent itemsets in two database scans. Toivonen [Toi96] proposes a sampling-based algorithm that requires at most two single database passes to determine all the frequent itemsets. It however makes a tradeoff between efficiency and accuracy and can generate more candidate itemsets than required. The DHP (Dynamic Hashing and Pruning) algorithm proposed by Park et al. [PCY95] uses a hash-based

technique to substantially reduce the number of candidate k-itemsets examined, for k >1. The ECLAT (Equivalence CLASS Transformation) algorithm proposed by Zaki [Zak00] mines frequent itemsets using a vertical data format – here, transaction identifiers (TIDs) are associated with each itemset, indicating the transactions in which the itemset is contained. Other notable contributions in this regard have been made by Holsheimer et al. [HKMT95], Savasere et al. [SON95] and Shenoy et al. [SHSB$^+$00]. A significant improvement over the Apriori algorithm is achieved by a method called FP-growth (Frequent Pattern Growth) proposed by Han et al. [HPY00]. This method differs from Apriori in that it mines all the frequent itemsets without candidate generation. [AAP01, GZ03b] are some significant alternatives/extensions to the FP-growth approach of mining frequent itemsets. Some notable contributions to the problem of incremental mining of frequent itemsets when transactions are added intermittently to a database are made in [CYNW96, AFLM99, DZ05, Dud09]. [AY98, HGN00, HCXY07] are some useful survey articles on frequent itemset generation algorithms.

A major challenge in frequent itemset mining is to handle the potentially large number of frequent itemsets that are extracted, specially when the value of *minsup* (the minimum support threshold) is low. All the subsets of a frequent itemset are frequent as well and so, a frequent itemset itself contains an exponential number of smaller, frequent sub-itemsets. Instead of enumerating every single frequent itemset, it is usually more useful to identify a small representative set of frequent itemsets, from which all other frequent itemsets can be derived. To this end, the notion of maximal frequent itemsets was introduced. A frequent itemset is maximal if it is not properly contained in any other frequent itemset. Every frequent itemset is contained in some maximal frequent itemset

and as such, the set of maximal frequent itemsets inherently and concisely contains all the frequent itemsets. A pioneering work for extracting maximal frequent itemsets was done by Bayardo [Bay98]. The algorithm Max-Miner proposed in [Bay98] uses an Apriori-based level-wise breadth-first traversal of the search space and also uses a lookahead pruning strategy to narrow the search and reduce the number of database scans. PMM (Parallel Max-Miner) proposed by Chung and Luo [CL03] is a parallel version of the sequential Max-Miner algorithm [Bay98] for shared-nothing multiprocessor systems. Concurrent to [CL03], Lin and Kedem [LK98, LK02] proposed the Pincer Search algorithm, which finds the set of maximal frequent sets by incorporating a bi-directional search. The algorithm maintains a superset of the set of maximal frequent sets and uses the top-down and bottom-up breadth first search strategies in conjunction to eliminate non-maximal frequent sets early on in the search, resulting in fewer database scans. MaxEclat and MaxClique are two algorithms proposed by Zaki [Zak00] to mine maximal patterns. Both these algorithms are based on graph-theoretic approaches. They divide the subset lattice into smaller pieces and these are then mined in a bottom-up Apriori-based manner using a vertical data format. The DepthProject algorithm proposed by Agarwal et al. [AAP00] finds long itemsets using a depth-first search of a lexicographic tree of itemsets. Here, the database is represented as a bitmap – each row in the bitmap is a bitvector corresponding to a transaction and each column corresponds to an item. The algorithm returns a superset of the maximal frequent itemsets and a post-pruning step is required to eliminate the non-maximal patterns. In [BCG01, BCFG+05], Burdick et al. extend the idea in DepthProject and present an algorithm called MAFIA, which improves counting efficiency by using vertical bitmaps to compress the transaction id (TID) list. Like

DepthProject, MAFIA also returns a superset of the maximal frequent itemsets and a post-pruning step is needed to remove the non-maximal patterns. A comprehensive performance study of the MAFIA algorithm is presented in [BCFG+03]. GenMax proposed by Gouda and Zaki [GZ01] is a backtrack search based algorithm that finds maximal frequent itemsets by using a number of optimizations to narrow the search. Unlike DepthProject and MAFIA, GenMax enumerates the exact set of maximal frequent itemsets. SmartMiner proposed by Zou et al. [ZCL02] uses an augmented dynamic reordering heuristic to determine the search path while mining maximal frequent itemsets. Compared with MAFIA and GenMax, SmartMiner generates a relatively smaller search tree and yields an order of magnitude improvement in speed. All-MFS is a randomized algorithm proposed by Gunopulos et al. [GKMS+03], which identifies maximal frequent itemsets in memory-resident datasets by iteratively attempting to extend a working pattern till failure. However, only a randomized version of the algorithm using vertical bit-vectors was studied and it does not guarantee that every maximal frequent itemset will be returned. Grahne and Zhu [GZ03a] extend the FP-growth method [HPY00] for determining frequent itemsets and present an algorithm called FPmax to mine only the set of maximal frequent itemsets. Then in [GZ03b], they extend the FPmax algorithm and present FPmax*, in which they use a variant of the FP-tree structure and a number of optimizations to further reduce the running time. In [GZ04a], Grahne and Zhu describe two approaches for reducing the main memory requirements of FPmax*. A theoretical analysis of the worst-case time-complexity of maximal frequent itemset mining is presented in [Yan04]. In [Yan04], the problem of mining maximal frequent itemsets is shown to be NP-hard.

The support of an itemset is the number of transactions in which the itemset appears. Though the set of maximal frequent itemsets provides a minimal representation for all frequent itemsets, it cannot be used to obtain the support information of the itemsets. The notion of closed frequent itemsets helps to alleviate this problem. An itemset s is said to be a closed frequent itemset if it is frequent and if none of the proper supersets of s have the same support count as that of s. The set of closed frequent itemsets provides a concise representation for all the frequent itemsets and can also be used to obtain the support count of each frequent itemset. Pasquier et al. [PBTL99a, PBTL99b] proposed a pioneering algorithm called A-Close for mining closed frequent itemsets and their support in a database. CLOSET proposed by Pei et al. [PHM00] extends the framework of the efficient frequent pattern mining method FP-growth (Frequent Pattern growth) [HPY00] and introduces the use of a FP-tree structure (which is a compressed representation of all the transactions in the database) in the closed frequent itemset discovery process. CLOSET uses a recursive divide-and-conquer and database projection approach to reduce the search space and quickly discover the closed frequent itemsets. Some significant improvements/alternatives to CLOSET, while respecting it's driving idea, were proposed by Wang et al. [WHP03], Qiu and Lan [QL06], Grahne and Zhu [GZ03b] and Liu et al. [LLY+03]. Grahne and Zhu [GZ04a] describe two approaches for reducing the main memory requirements of the FPclose algorithm [GZ03b]. Burdick et al. [BCGF+05] show how the well-known MAFIA algorithm, primarily intended for maximal pattern mining can be extended to discover closed frequent itemsets. CHARM, another well-known algorithm for enumerating closed frequent itemsets, proposed by Zaki and Hsiao [ZH02], uses a novel IT-tree (itemset-tidset tree) structure to simultaneously explore both

the itemset and transaction space. CHARM utilizes a hash-based approach to eliminate non-closed itemsets and a novel vertical data representation called *diffset* for fast support computations. Some improvements of this algorithm were proposed, mainly the DCI-CLOSED algorithm [LOP04, LOP06] and the LCM (Linear time Closed itemset Miner) algorithm [UAUA03, UAUA04, UKA04]. Singh et al. [SSM05, SSMP06] propose an algorithm called CloseMiner which discovers the set of closed frequent itemsets by grouping the set of itemsets into non-overlapping clusters. Song et al. [SYX08] introduce an index array and use it to determine the set of closed frequent itemsets. Yahia et al. [YHN06] present an analytical and structural survey of various closed frequent itemset mining algorithms. An in-depth performance analysis of the different algorithms, based on memory consumption and the advantages and/or limitations of optimization strategies used, is provided.

Two workshops – FIMI (Frequent Itemset Mining Implementations)'03 and FIMI'04 – were held at ICDM (IEEE International Conference on Data Mining)-2003 and ICDM-2004 respectively to determine the best algorithms for mining frequent itemsets, maximal frequent itemsets and closed frequent itemsets. The source codes of all the implementations accepted at these two workshops along with several datasets for benchmarking purposes are available at http://fimi.cs.helsinki.fi/ . The proceedings of FIMI'03 and FIMI'04 are available at http://www.ceur-ws.org/Vol-90/ and http://ceur-ws.org/Vol-126/ respectively. [GZ04b] is a report on FIMI'03.

Because of their combinatorial explosion, the tasks of mining frequent itemsets, maximal frequent itemsets and closed frequent itemsets are not trivial and are typically NP-hard. The notion of k-frequent intervals was introduced by Yu [Yu02]. A non-empty

interval is said to be k-frequent if it is contained in at least k intervals in a database of intervals. Yu proposed a data-structure called I-Tree in [Yu02] and a polynomial-time Apriori-based algorithm that uses this data-structure to determine the set of k-frequent intervals. In [Lin03], Lin introduces the notion of maximal k-frequent intervals. A non-empty interval is said to be maximal k-frequent if it is k-frequent and not properly contained in another k-frequent interval. The set of maximal k-frequent intervals is the most compact representation of the entire set of k-frequent intervals. Lin [Lin03] presents a polynomial time algorithm called PT (Preorder Traversal) that uses the I-Tree data-structure proposed by Yu [Yu02] to determine the set of maximal k-frequent intervals. To the best of our knowledge, no other work on the problem of mining maximal k-frequent intervals exists in the public domain.

Analogous to closed frequent itemsets, it is possible to introduce the notion of closed k-frequent intervals. A closed k-frequent interval is a non-empty interval that is k-frequent and whose absolute support value (i.e. the number of intervals that contain it) is different from that of every interval in which it is properly contained. To the best of our knowledge, no earlier work on the problem of mining closed k-frequent intervals exists in the public domain.

Some pioneering work related to the problem of finding calendar-based periodicities of events/patterns that occur only during certain time-intervals, has been done in the context of a time-stamped transactional database. In a time-stamped transactional database, each transaction in the database contains the time at which the transaction has occurred. In such a database, it is possible to find association rules (viz. relationships among items in the database) that hold only during certain time-intervals. The problem of mining

cyclic patterns viz. association rules that hold periodically over time, in a time-stamped transactional database is addressed by Ozden et al.[ORS98]. However, the model used by Ozden et al. [ORS98] has limited expressiveness – it uses only a single granularity unit to describe time. The methods proposed in [ORS98] can only identify patterns that repeat after constant single granularity time units – such as patterns that occur every 12 hours, every 7 days etc. In such an approach, many commonly encountered variations – e.g. patterns that occur on the first day of every month or on the first business day of every month etc – get missed because the distance between the first days of two consecutive months, the distance between the first business days of two consecutive months etc. is not constant. Moreover, the model used in [ORS98] fails to capture patterns that are *approximately* periodic. Ramaswamy et al. [RMS98] extend the work done in [ORS98] by proposing a method that can identify totally periodic as well as approximately periodic patterns (viz. association rules) in a time-stamped transactional database. Ramaswamy et al. [RMS98] introduce the usage of a *calendar algebra* that allows users to specify a calendar – viz. a set of time-intervals. Next, given a user-specified calendar, Ramaswamy et al. propose a method to mine *calendric* association rules – viz. association rules that hold either during all or during *enough* number of time-intervals in the user-specified calendar. Although a single granularity of time is used in [RMS98] also, calendric association rules discovered by Ramaswamy et al. [RMS98] offer far more expressive power than the cyclic association rules identified by Ozden et al. [ORS98]. In most cases, however, users lack prior knowledge about the time-intervals (viz. the calendar) that need to be specified in order to discover calendric association rules. Li et al. [LNWJ03] propose a method which requires relatively less prior knowledge to discover periodic patterns (viz.

rules) in a time-stamped transactional database. Li et al. [LNWJ03] use a *calendar-schema* defined by a hierarchy of calendar concepts – such as (year, month, day), (week, hour, day) etc – as a template and discover all association rules that are totally or approximately periodic with respect to the given calendar schema. For example, for the calendar-schema (year, month, day), association rules that hold – on every day of March 2007, on more than 70% of December days etc – could be discovered. As such, the approach used in [LNWJ03] not only requires less a priori knowledge than the method given by [RMS98], but it is also more flexible in the sense that a single calendar schema maps to several calendar-based periodicities – leading to the potential discovery of a relatively greater number of periodic patterns (rules) in a time-stamped transactional database. The approach of using a calendar schema as a template was also utilized by Li et al. [LWJ00] to determine both total as well as partial calendar-based periodicities of a pattern/event – under the assumption that the pattern/event together with the multiset of timestamps at which the pattern/event has occurred is known. Mahanta et al. [MMB08] consider patterns/events that occur during certain time-intervals. Here, it is assumed that the timestamps are expressed in the form of a hierarchy of calendar concepts – viz. year/month/day, week/day/hour etc. Given the time-intervals in which a pattern/event has occurred, a method is proposed by Mahanta et al. [MMB08] to extract periodicities of the pattern/event at the different levels of the timestamp hierarchy. For example, if the timestamps are in the form of year/month/day, then yearly and monthly periodicities of the pattern/event can be discovered. The method proposed by [MMB08] is able to detect both partial as well as total periodicities of the event/pattern under study.