Towards Generic Pattern Mining*
(Extended Abstract)

Mohammed J. Zaki, Nilanjana De, Feng Gao, Nagender Parimi,
Benjarath Phoophakdee, Joe Urban Vineet Chaoji,
Mohammad Al Hasan, and Saeed Salem**

Computer Science Department, Rensselaer Polytechnic Institute,
Troy NY 12180

1 Introduction

Frequent Pattern Mining (FPM) is a very powerful paradigm which encompasses an entire class of data mining tasks. The specific tasks encompassed by FPM include the mining of increasingly complex and informative patterns, in complex structured and unstructured relational datasets, such as: Itemsets or co-occurrences [1] (transactional, unordered data), Sequences [2,8] (temporal or positional data, as in text mining, bioinformatics), Tree patterns [9] (XML/semistructured data), and Graph patterns [4,5,6] (complex relational data, bioinformatics). Figure 1 shows examples of these different types of patterns; in a generic sense a pattern denotes links/relationships between several objects of interest. The objects are denoted as nodes, and the links as edges. Patterns can have multiple labels, denoting various attributes, on both the nodes and edges.

We have developed the Data Mining Template Library (DMTL) [10], a generic collection of algorithms and persistent data structures for FPM, which follows a generic programming paradigm[3]. DMTL provides a systematic solution for the whole class of pattern mining tasks in massive, relational datasets. DMTL allows for the isolation of generic containers which hold various pattern types from the actual mining algorithms which operate upon them. We define generic data structures to handle various pattern types like itemsets, sequences, trees and graphs, and outline the design and implementation of generic data mining algorithms for FPM, such as depth-first and breadth-first search. It provides persistent data structures for supporting efficient pattern frequency computations using a tightly coupled database (DBMS) approach. One of the main attractions of a generic paradigm is that the generic algorithms for mining are guaranteed to work for **any** pattern type. Each pattern is characterized by inherent properties that it satisfies, and the generic algorithm exploits these properties to perform the mining task efficiently. Full details of the DMTL approach appear in [10]. Here we selectively highlight its main features.

* This work was supported in part by NSF CAREER Award IIS-0092978, DOE Career Award DE-FG02-02ER25538, and NSF grants EIA-0103708 and EMT-0432098.
** We thank Paolo Palmerini and Jeevan Pathuri for their previous work on DMTL.

S.K. Pal et al. (Eds.): PReMI 2005, LNCS 3776, pp. 91–97, 2005.

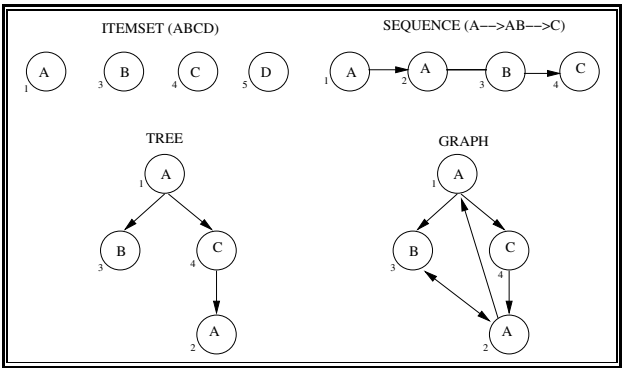


Fig. 1. FPM Instances

2 DMTL: Data Structures and Algorithms

DMTL is a collection of generic data mining algorithms and data structures. In addition, DMTL provides persistent data and index structures for efficiently mining any type of pattern or model of interest. The user can mine custom pattern types, by simply defining the new pattern types, but the user need not implement a new algorithm - the generic DMTL algorithms can be used to mine them. Since the mined models and patterns are persistent and indexed, this means the mining can be done efficiently over massive databases, and mined results can be retrieved later from the persistent store.

Containers: Figure 2 shows the different DMTL container classes and the relationship among them. At the lowest level are the different kinds of pattern-types one might be interested in mining. A pattern is a generic container instanti-

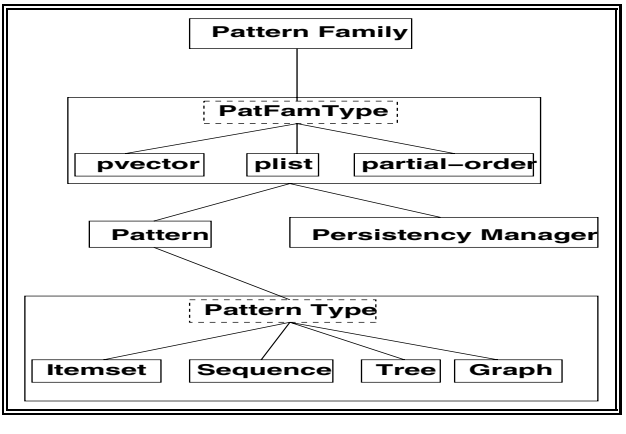


Fig. 2. DMTL Container Hierarchy

ated for one of the pattern-types. There are several pattern family types (such as pvector, plist, etc.) which together with a persistency manager class make up different pattern family classes. In DMTL a pattern is a generic container, which can be instantiated as an itemset, sequence, tree or a graph, specified as `Pattern<class P>` by means of a template argument called Pattern-Type (`P`). A generic pattern is simply a Pattern-Type whose frequency we need to determine in a larger collection or database of patterns of the same type. A pattern type is the specific pattern to be mined, e.g. itemset, and in that sense is not a generic container. DMTL has the itemset, sequence, tree and graph pattern-types defined internally; however the users are free to define their own pattern types, so long as the user defined class provides implementations for the methods required by the generic containers and algorithms. In addition to the basic pattern classes, most pattern mining algorithms operate on a collection of patterns. The pattern family is a generic container `PatternFamily <class PatFamType>` to store groups of patterns, specified by the template parameter `PatFamType`. `PatFamType` represents a persistent class provided by DMTL, that provides seamless access to the members, whether they be in memory or on disk. This class provides the required persistency in storage and retrieval of patterns. DMTL provides several pattern family types to store groups of patterns. Each such class is templatized on the pattern-type (P) and a persistency manager class `PM`. An example is `pvector <class P, class PM>`, a persistent vector class. It has the same semantics as a STL vector with added memory management and persistency. Another class is `plist<P,PM>`. Instead of organizing the patterns in a linear structure like a vector or list, another persistent family type DMTL class, `partial-order <P,PM>`, organizes the patterns according to the sub-pattern/super-pattern relationship.

Algorithms: The pattern mining task can be viewed as a search over the pattern space looking for those patterns that match the minimum support constraint. For instance in itemset mining, the search space is the set of all possible subsets of items. Within DMTL we attempt to provide a unifying framework for the wide range of mining algorithms that exist today. DMTL provides generic algorithms which by their definition can work on any type of pattern: Itemset, Sequence, Tree or Graph. Several variants of pattern search strategies exist in the literature, depth-first search (DFS) and breadth-first search (BFS) being the primary ones. BFS has the advantage of providing better pruning of candidates but suffers from the cost of storing all of a given level's frequent patterns in memory. Recent algorithms for mining complex patterns like trees and graphs have focused on the DFS approach, hence it is the preferred choice for our toolkit as well. Nevertheless, support for BFS mining of itemsets and sequences is provided.

3 DMTL: Persistency and Database Support

DMTL employs a back-end storage manager that provides the persistency and indexing support for both the patterns and the database. It supports DMTL by seamlessly providing support for memory management, data layout, high-

performance I/O, as well as tight integration with database management systems (DBMS). It supports multiple back-end storage schemes including flat files, embedded databases, and relational or object-relational DBMS. DMTL also provides persistent pattern management facilities, i.e., mined patterns can themselves be stored in a pattern database for retrieval and interactive exploration.

Vertical Attribute Tables. To provide native database support for objects in the vertical format, DMTL adopts a fine grained data model, where records are stored as *Vertical Attribute Tables* (VATs). Given a database of objects, where each object is characterized by a set of properties or attributes, a VAT is essentially the collection of objects that share the same values for the attributes. For example, for a relational table, `cars`, with the two attributes, `color` and `brand`, a VAT for the property `color=red` stores all the transaction identifiers of cars whose color is red. The main advantage of VATs is that they allow for optimizations of query intensive applications like data mining where only a subset of the attributes need to be processed during each query. These kinds of vertical representations have proved to be useful in many data mining tasks [7,8,9]. In DMTL there is one VAT per pattern-type. Depending on the pattern type being mined the vat-type class may be different. Accordingly, their intersection shall vary as well.

Storage & Persistency Manager. The database support for VATs and for the horizontal family of patterns is provided by DMTL in terms of the following classes, which are illustrated in Figure 3. `Vat-type` is a class describing the vat-type that composes the body of a VAT, for instance `int` for itemsets and `pair<int,time>` for sequences. `VAT<class V>` is the class that represents VATs. This class is composed of a collection of records of vat-type V. `Storage<class PM>` is the generic persistency-manager class that implements the physical persistency for VATs and

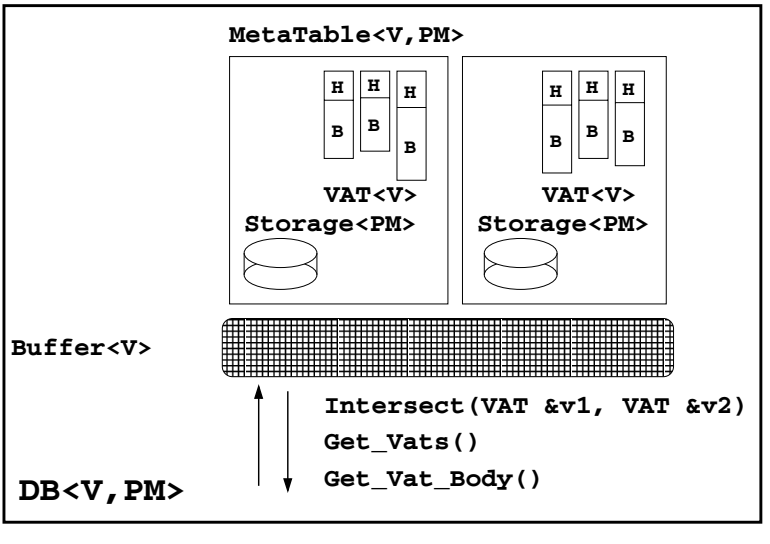


Fig. 3. DMTL: High level overview of the different classes used for Persistency

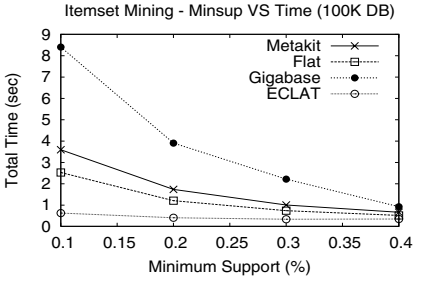
other classes. The class PM provides the actual implementations of the generic operations required by Storage. For example, PM_metakit and PM_gigabase are two actual implementations of the `Storage` class in terms of different DBMS like Metakit (`http://www.equi4.com/metakit/`), a persistent C++ library that natively supports the vertical format, and Gigabase (`http://sourceforge.net/projects/gigabase`), an object-relational database. Other implementations can easily be added as long as they provide the required functionality.

`Buffer<class V>` provides a fixed-size main-memory buffer to which VATs are written and from which VATs are accessed, used for buffer management to provide seamless support for main-memory and out-of-core VATs (of type V). `MetaTable<class V, class PM>` represents a collection of VATs. It stores a list of VAT pointers and the adequate data structures to handle efficient search for a specific VAT in the collection. It also provides physical storage for VATs. It is templatized on the vat-type V and on the `Storage` implementation PM. In the figure the H refers to a pattern and B its corresponding VAT. The `Storage` class provides for efficient lookup of a particular `VAT` object given the header. `DB<class V, class PM>` is the database class which holds a collection of `Metatables`. This is the main user interface to VATs and constitutes the database class DB referred to in previous sections.

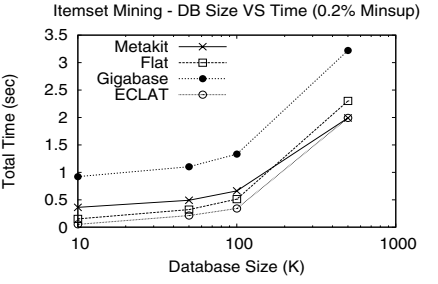
4 Experiments

Templates provide a clean means of implementing our concepts of genericity of containers and algorithms; hence DMTL is implemented using the C++ Standard Template Library [3]. We present some experimental results on different types of pattern mining. We used the IBM synthetic database generator [1] for itemset and sequence mining, the tree generator from [9] for tree mining and the graph generator by [5], with sizes ranging from $10K$ to $500K$ (or 0.5 million) objects. The experiment were run on a Pentium4 2.8Ghz Processor with 6GB of memory, running Linux.

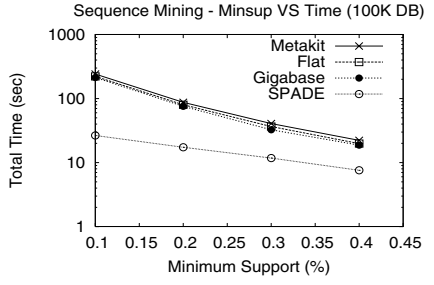
Figure 4 shows the DMTL mining time versus the specialized algorithms for itemset mining (ECLAT [7]), sequences (SPADE [8]), trees (TreeMiner [9]) and graphs (gSpan [6]). For the DMTL algorithms, we show the time with different persistency managers/databases: flat-file (Flat), metakit backend (Metakit) and the gigabase backend (Gigabase). The left hand column shows the effect of minimum support on the mining time for the various patterns, the column on the right hand size shows the effect of increasing database sizes on these algorithms. Figures 4(a) and 4(b) contrast performance of DMTL with ECLAT over varying supports and database sizes, respectively. As can be seen in, Figure 4(b), DMTL(Metakit) is as fast as the specialized algorithm for larger database sizes. Tree mining in DMTL (figures 4(e) and 4(f)) substantially outperforms TreeMiner; we attribute this to the initial overhead that TreeMiner incurs by reading the database in horizontal format, and then converting it into the vertical one. For graph and sequence patterns, we find that DMTL is at most, within a factor of 10 as compared to specialized algorithms and often much closer (Figure 4(d)).



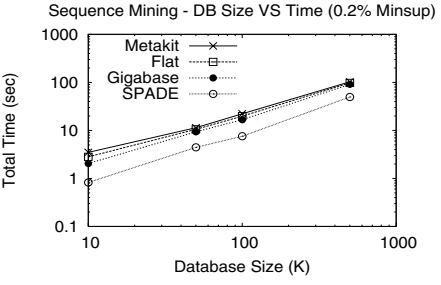
(a) Itemsets:Varying Minsup



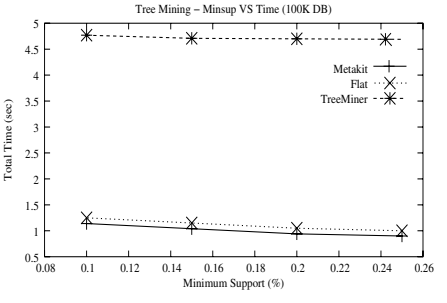
(b) Itemsets: Varying DB size



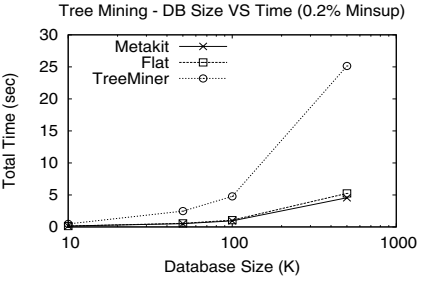
(c) Sequences: Varying Minsup



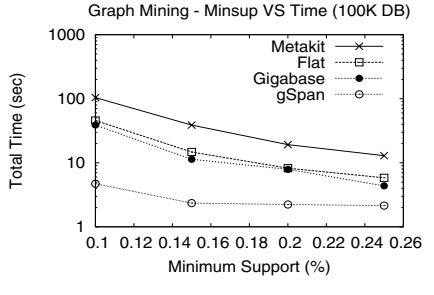
(d) Sequences: Varying DB size



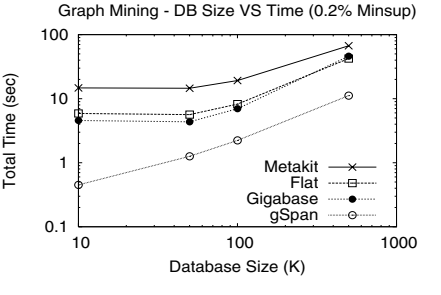
(e) Trees: Varying Minsup



(f) Trees: Varying DB size



(g) Graphs: Varying Minsup



(h) Graphs: Varying DB size

Fig. 4. Performance: Itemset, Sequence, Tree and Graph Mining

Overall, the timings demonstrate that the performance and scalability benefits of DMTL are clearly evident with large databases. For itemsets, our experiments reported that ECLAT breaks for a database with 5 million records, while DMTL terminated in 23.5s with complete results.

5 Conclusions

The generic paradigm of DMTL is a first-of-its-kind in data mining, and we plan to use insights gained to extend DMTL to other common mining tasks like classification, clustering, deviation detection, and so on. Eventually, DMTL will house the tightly-integrated and optimized primitive, generic operations, which serve as the building blocks of more complex mining algorithms. The primitive operations will serve all steps of the mining process, i.e., pre-processing of data, mining algorithms, and post-processing of patterns/models. Finally, we plan to release DMTL as part of open-source, and the feedback we receive will help drive more useful enhancements. We also hope that DMTL will provide a common platform for developing new algorithms, and that it will foster comparison among the multitude of existing algorithms. For more details on DMTL see [10].

References

1. R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. Inkeri Verkamo. Fast discovery of association rules. In U. Fayyad and et al, editors, *Advances in Knowledge Discovery and Data Mining*, pages pp. 307–328. AAAI Press, Menlo Park, CA, 1996.
2. R. Agrawal and R. Srikant. Mining sequential patterns. In *11th Intl. Conf. on Data Engg.*, 1995.
3. M. H. Austern. *Generic Programming and the STL.* Addison Wesley Longman, Inc., 1999.
4. A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *4th European Conference on Principles of Knowledge Discovery and Data Mining*, September 2000.
5. M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *1st IEEE Int'l Conf. on Data Mining*, November 2001.
6. X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *IEEE Int'l Conf. on Data Mining*, 2002.
7. M. J. Zaki. Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):372-390, May-June 2000.
8. M. J. Zaki. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning Journal*, 42(1/2):31–60, Jan/Feb 2001.
9. M. J. Zaki. Efficiently mining frequent trees in a forest. In *8th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining*, July 2002.
10. Mohammed J. Zaki, Nagender Parimi, Nilanjana De, Feng Gao, Benjarath Phoophakdee, Joe Urban, Vineet Chaoji, Mohammad Al Hasan, and Saeed Salem. Towards generic pattern mining. In *Int'l Conf. on Formal Concept Anaysis*, February 2005.