TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 15/20 pp719-728
Volume 12, Number 6, December 2007

Parallel Frequent Pattern Discovery: Challenges and Methodology*

ZHANG Yuzhou (张宇宙), WANG Jianyong (王建勇)**, ZHOU Lizhu (周立柱)

Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China

Abstract: Parallel frequent pattern discovery algorithms exploit parallel and distributed computing resources to relieve the sequential bottlenecks of current frequent pattern mining (FPM) algorithms. Thus, parallel FPM algorithms achieve better scalability and performance, so they are attracting much attention in the data mining research community. This paper presents a comprehensive survey of the state-of-the-art parallel and distributed frequent pattern mining algorithms with more emphasis on pattern discovery from complex data (e.g., sequences and graphs) on various platforms. A review of typical parallel FPM algorithms uncovers the major challenges, methodologies, and research problems in the field of parallel frequent pattern discovery, such as work-load balancing, finding good data layouts, and data decomposition. This survey also indicates a dramatic shift of the research interest in the field from the simple parallel frequent itemset mining on traditional parallel and distributed platforms to parallel pattern mining of more complex data on emerging architectures, such as multi-core systems and the increasingly mature grid infrastructure.

Key words: frequent pattern mining; parallel computing; dynamic load balancing

Introduction

Frequent patterns are prevalent in real-life data, such as sets of items bought together in a superstore and some molecular fragments frequently appearing in a certain class of molecules with similar functions[1]. Frequent pattern mining has been successfully applied to association rule mining, pattern-based classification and clustering, and finding correlated items, and has become an essential data mining task.

The problem of discovering frequent patterns can be stated as follows. Given a transaction database, D, which consists of a set of transactions, $\{t_1,t_2,\ldots,t_n\}$, a user-specified minimum support δ (δ is a real number and $0<\delta\leqslant 1$), the frequent pattern mining (FPM) problem is to discover the complete set of all patterns contained in at least a specified percentage, δ, of transactions in the transaction database. Depending on the specific problem formulation, the input transaction and pattern can be an itemset, a sequence, a tree, or a graph. This work mainly focuses on the three popular frequent pattern mining problem formulations, frequent itemset mining (FIM), frequent sequence mining (FSM), and frequent graph mining (FGM).

The search space of frequent pattern mining can be modeled as a lattice structure in which patterns are layered according to their size and the patterns are sorted according to a certain order at each layer. Figure 1 shows an example of a lattice formed by all subsets of itemset $I=\{1,2,3,4,5\}$. The dimensionality of a pattern lattice is always exponential to the problem size. For example, for an itemset transaction database with n distinct items, the number of possible patterns is 2^n. For a single-item event sequence database with n distinct items and a maximum transaction length of m, the number of all possible patterns is $(n+n^2+n^3+\cdots+n^m)$.

Received: 2006-11-30; revised: 2007-07-12

* Supported by the Basic Research Foundation of Tsinghua National Laboratory for Information Science and Technology (TNList)

** To whom correspondence should be addressed.
E-mail: jianyong@tsinghua.edu.cn; Tel: 86-10-62789150

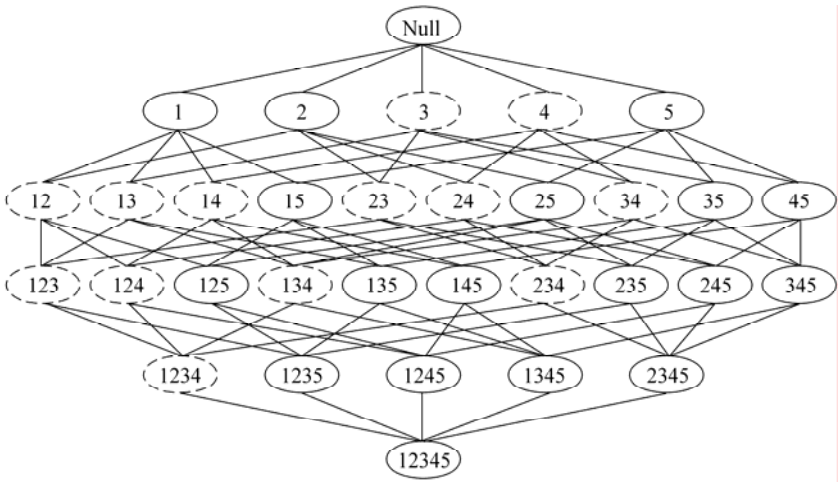

Fig. 1 Lattice of subsets of {1,2,3,4,5} (Note that if {1,2,3,4} is frequent, all its sub-itemsets are frequent, as shown by the dashed circles)

The huge search space makes frequent pattern mining on large databases a very computationally intensive problem. The performance of a serial FPM algorithm is limited by its single processor computing capability and finite memory space; thus, they do not scale to large datasets. Therefore, it is very necessary to exploit high performance parallel/distributed computing to relieve the current FPM algorithms from the sequential bottleneck, and helps the FPM algorithms scale to massive datasets.

Understanding the properties of pattern lattice and common pattern enumeration techniques is the key to designing serial FPM algorithms and to exploring their parallelizability.

All serial FPM algorithms adopt a specific lattice traversal paradigm so that all frequent patterns can be enumerated without redundancy and incompleteness and so that some unnecessary parts of the lattice can be bypassed without hurting the integrity. For example, the Apriori algorithm[2] adopts a level-wise method, FP-growth[3] divides the pattern space into separate spaces with respect to the item appearance, SPADE[4] uses a clever suffix-based intra equivalent class level-wise method which preserves a tree-like pattern arrangement, and TreeProjection[5] also arranges the pattern lattice into a tree form based on prefix. A well-designed algorithm usually adopts some kind of pruning techniques to avoid traversing unpromising parts of the search space. The famous Apriori property often used to prune some futile sub-search spaces in FPM algorithms says that no super pattern of an infrequent pattern can be frequent.

Although many efficient serial FPM algorithms have been proposed in recent years with a wide variety of implementation details, they can be differentiated by their pattern lattice enumeration strategies, support counting methods, and search space pruning techniques. In addition, some algorithms use special data structures like hash table to achieve better performance.

There are two basic classes of frequent pattern mining algorithms with respect to the pattern space enumeration paradigm. The first one uses the level-wise candidate-generation-and-test (CGT) method. Representative algorithms in this class include Apriori[2], GSP[6], and AGM[7], which correspond to frequent itemset, frequent sequence, and frequent graph mining problems, respectively. In this class of algorithms, the set of all already mined frequent k-patterns (i.e., frequent patterns of length k) is self-joined to generate the set of (k+1)-pattern candidates which will be further tested against the database to get their supports and to filter out patterns infrequent. This process goes on iteratively level by level until longer patterns cannot be generated.

The other important class of frequent pattern mining algorithms is based on the depth-first pattern growth and database projection (PGDP) method. Representative algorithms in this class are FP-growth[3] for itemset mining, PrefixSpan[8] for sequence mining, and gSpan[9] for graph mining. In this class of algorithms, the k-pattern currently being mined is extended by one item, usually in a predefined order, to form the (k+1)-

pattern. A projection database will be generated in each step during the pattern growth. The algorithm proceeds recursively in a depth-first manner until the entire pattern tree has been traversed.

1 Challenges in Parallel Pattern Discovery

Multiple processors can be used to solve frequent pattern mining problems with the expected speedup being almost equal to the number of processors, but several issues have to be addressed due to the characteristics of frequent pattern mining problems.

In order to find the support of a pattern, a compare-and-count operation has to be done against all transactions in the database. A naive task parallelism would evenly distribute the transactions among the processors. While mining a pattern, each processor performs the compare-and-count operation on its local set of transactions and with a sum reduction to get the global pattern support. This is the so-called count-distribution (CD) method. The problem with this method is its excessively fine granularity of task decomposition, which usually results in endless communications and synchronization that make the count-distribution method impractical.

A frequent pattern must appear in at least a certain number of database transactions. A transaction usually contains many patterns, and therefore, contributes to the counts in many frequent patterns. Consider an undirected bipartite graph $G=(V_t, V_p, E_c)$, where V_t represents the set of transactions in the database and V_p corresponds to the set of frequent patterns. There exists an edge $(u,v)\in E_c$ where $u\in V_t$ and $v\in V_p$ if transaction u contains pattern v. Figure 2 shows an example of a bipartite graph for a sequential database. The two parts of the bipartite graph are densely connected, which implies tight coupling between the transactions and patterns in terms of the containment relationship.

The aggregate pattern discovery task could be distributed among processors by splitting the pattern lattice with each processor responsible for mining the subset of patterns assigned to it. The problem here is that because of the tight coupling between transactions and patterns, almost all processors have to access almost all the transactions in the database. For a non-uniform memory access architecture (NUMA) system, the database should be replicated on each processor; otherwise, access overhead to non-local transactions can be overwhelming. However, database replication impairs the advantage of the large aggregate memory and storage space on parallel platforms. An improved method is to selectively replicate the database according to the need, but this requires additional but affordable statistic process on the database.

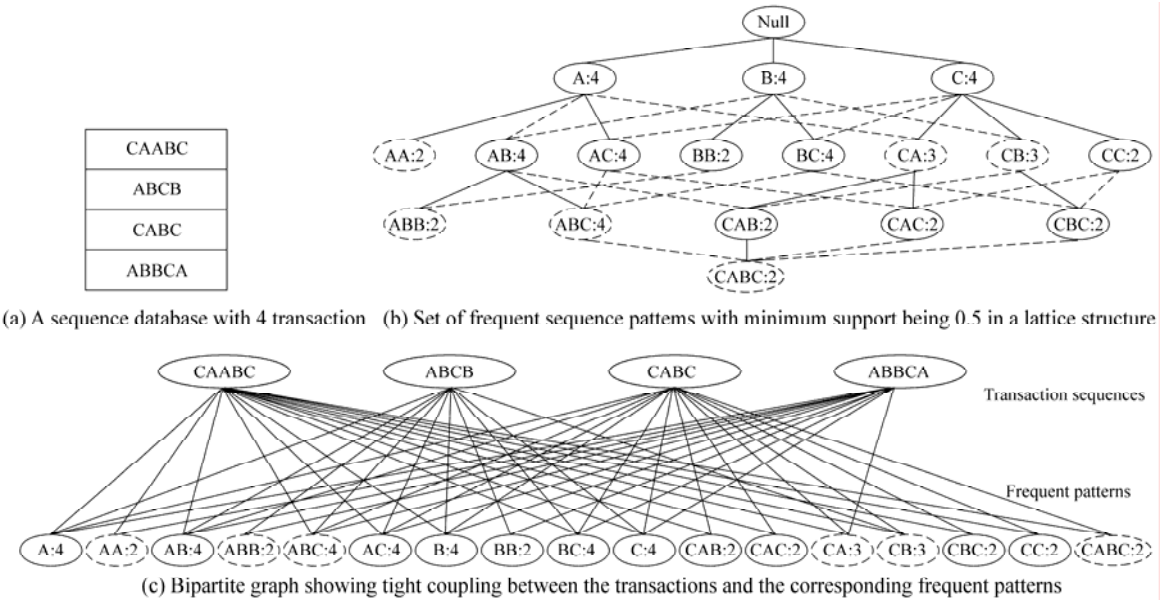

Fig. 2 Tight coupling between the transactions and patterns (The dashed lines indicate prefix relations and the dashed circles indicate closed patterns)

Load balancing is of great significance for achieving good speed up for any parallel algorithm. The difficulty in designing a static load balancing strategy for the parallel FPM algorithms is that a method is required to approximate the relative mining time of the task unit at the proper level of granularity. Afterwards, some task assignment strategy (e.g., greedy or bin-packing algorithm) can be devised to assign work to processors in a balanced way. However, in most datasets the pattern tree is unbalanced and the pattern-transaction bipartite graph is skewed; thus, the workload approximation can only be heuristic. A major challenge is to estimate the running time for discovering a subset of frequent patterns as accurate as possible, while considering the pattern enumeration methods of the counterpart serial algorithms.

Dynamic load balancing is expected to eliminate the difficulty of approximating the pattern mining task. A parallel FPM algorithm must be able to partition the total pattern discovery task to produce fine granulated or recursively able-partitioned tasks. In this way tasks can be split and transferred between processors. pSPADE[10] is a good example. Another problem is to minimize the overhead introduced by task transfers, especially when accompanied by massive transaction transfers or when the architecture has a slow network like the grid environment.

For fast execution on large datasets, all serial FPM algorithms adopt sophisticated pattern space pruning techniques and compact data structures. Parallelization of these FPM algorithms needs to preserve the validity and efficiency of the detailed techniques and structures. As an example, some pruning strategies need to refer to already mined patterns which probably were mined or maintained by another processor. Additional message passing may be inescapable. Generally, pattern space pruning techniques always bring additional costs and after parallelization the costs may be nonlinearly amplified. Therefore, the parallel FPM algorithm designer must be very cautious about tradeoffs among these factors.

2 Algorithms

This section reviews state-of-the-art, representative parallel FPM algorithms. The subsections are organized according to their problem formulations, that is, itemset mining, sequence mining, and graph mining.

2.1 Parallel frequent itemset mining

Frequent itemset mining (FIM, also known as association rule mining) is the simplest one in the FPM problem family. For the FIM problem, there is a set of items, I, and a database, D, consisting of a set of transactions, each of which is a subset of I.

The parallelizing association rule mining algorithms has attracted great interest for a decade. Zaki[11] presented an excellent survey of parallel and distributed association rule mining algorithms. However, all of the surveyed parallel algorithms were based on the old-fashioned Apriori-like frequent itemset mining algorithms, which suffered from the inherent problem of the CGT paradigm. This paper will not repeat Zaki's work[11], but will only summarize the basic methodologies for parallel FIM algorithms of the CGT class.

(1) Count distribution First, partition the database evenly and replicate the k-patterns fully over all processors. Next, each processor independently generates its complete set of candidate (k+1)-patterns and counts them over its local database. Finally, a sum reduction is performed to get the global count. This process goes on in level-wise.

(2) Data distribution The database is also partitioned among the processors. Each processor generates a disjoint set of candidates. Because each processor needs the entire database to count the candidates, communication among the processors is inevitable.

(3) Hybrid methods.

Jin and Agrawal[12] implemented an Apriori-based association rule mining algorithm running on a cluster of SMPs. The distributed memory parallelization strategy was the same as an earlier algorithm called CD. The shared memory parallelization strategy is similar to another algorithm called CCPD. Both have been surveyed by Ref. [11], and therefore, the details will not be discussed here. This implementation is based on a general-purpose middleware designed for parallel data mining which can process disk resident datasets.

In most parallel FIM algorithms adopting the CGT framework, all processors must first accomplish the candidate generation and test work for the current level by intensive cooperation with each other before they can go together to the next pattern length level. Therefore, a level-wise barrier is required. The cascade running mode results in an enormous amount of synchronization overhead and makes it hard to conduct a

dynamic load balancing strategy. Zaki's survey[11] showed that none of reviewed algorithms adopts a dynamic load balancing strategy.

FP-growth[3] is a state of the art frequent itemset mining algorithm which uses the PGDP paradigm. A compact data structure called FP-tree is first built to hold all the information in the original database. Then, a divide-and-conquer search is performed on the read-only FP-tree structure in a bottom-up fashion. To illustrate the divide-and-conquer method in FP-growth which is the key for understanding how to parallelize it, consider a transaction database in which I={A:7,B:6, C:4}. The number after the colon is the support of the corresponding item. FP-growth first mines patterns containing item C, then patterns containing B but no C, finally patterns containing item A but no B nor C.

MLFPT[13] is a simple parallel algorithm based on FP-growth for mining frequent itemsets on shared memory machines. The FP-tree building phase uses a data parallel paradigm, in which each processor builds a local FP-tree structure using its assigned part of the transactions. The FP-tree structures share a global header table through which every processor can access the whole structure. The frequent items in the header table are assigned among processors using their support as estimation of the relative mining time. Each processor can then independently mine patterns involving each assigned frequent item. The key idea behind this task parallel paradigm is the divide-and-conquer nature of the FP-growth algorithm.

Later, Javed and Khokhar[14] proposed another FP-growth-based parallel algorithm called PFP-tree but running on a distributed memory architecture. The FP-tree building process is similar to that in MLFPT except that multiple FP-trees reside in different physical memory spaces. The mining task is also divided in the same way as in MLFPT. Another difference is that the conditional pattern base of each 1-itemset must be exchanged among processors so that each can hold the needed portion of the database locally. The common drawback of MLFPT and PFP-tree algorithms is their coarse estimates of the relative mining time.

The authors of MLFPT[13] later presented a new method[15] called the Inverted Matrix for mining frequent itemsets on distributed memory architectures. The transaction database is first reorganized into a layout called the inverted matrix. This matrix structure is fully replicated among all the processor nodes. As with MLFPT, the frequent items are assigned among the processors. When an item is assigned to a processor, the processor is then responsible for mining patterns containing only this item and items having a larger rank in the item frequency ascending order. After the item assignment process, each processor builds a COFI-tree (co-occurrence frequent item tree) structure for each of its assigned items. Finally, a non-recursive mining is performed on the COFI-tree. An interesting observation is that, due to the support ascending order of the frequent items in the inverted matrix structure, an item with a high support usually corresponds to a small COFI-tree, which guarantees good load balance.

The intrinsic divide-and-conquer nature of FP-growth can easily be parallelized. MLFPT and PFP-tree both utilize a one-time task assignment paradigm by dividing the FP-tree header table. Although the inverted matrix algorithm has no serial counterpart, it is very similar to FP-growth. The static load balancing strategy of the Inverted Matrix algorithm is more reasonable than MLFPT and PFP-tree.

2.2 Parallel frequent sequence mining

Frequent sequence mining (FSM) algorithms mine patterns with consideration of the temporal order between items. Assume there is a set of items I={A,B,C,D}, a sequential transaction is in a similar form to the example <(C)(AB)(AC)>. Simply speaking, a sequence is a list of temporally ordered itemsets (also called events).

The methodologies adopted in most serial FSM algorithms are analogies of existing FIM algorithms. However, mining sequential patterns is a greater challenge because the special or temporal order between events has to be considered. The large combinational number of subsequences makes the sequence pattern lattice more branchy and skewed than the itemset pattern lattice. While parallel frequent itemset mining has attracted tremendous interest, there is relatively little work on parallel frequent sequence mining.

Shintani and Kitsuregawa[16] proposed three parallel algorithms for distributed memory computer based on GSP[4], which is a classic CGT style FSM algorithm. All three algorithms partition the database among processor nodes. The NPSPM algorithm and the SPSPM algorithm are merely analogies of the Counter Distribution and Data Distribution[17] algorithm

designed for parallel FIM problems, respectively. HPSPM is similar to SPSPM but utilizes a more intelligent strategy to partition the candidate sequences among processors using a hashing mechanism. HPSPM also reduces the amount of communication overhead needed to count the global support. The target machine in both Agrawal and Shafer[17] and Shintani and Kitsuregawa[16] was an IBM SP2 distributed memory machine.

Guralnik and Karypis[5] designed a variant of the parallel tree-projection-based frequent sequence mining algorithm also targeting distributed memory architecture. Their data parallel formulation adopts the classic count-distribution method, where each processor redundantly builds the same pattern tree as in the serial algorithm. After the data parallel formulation runs on the first k levels of the pattern tree, the algorithm switches to task parallel mode. As in other parallelized PGDP pattern mining algorithms, the pattern tree node is the unit task for parallelizing. Nodes at the k-level are distributed among processors which then independently proceed to generate sub-forests rooted at the assigned nodes. Processors must exchange part of their local databases before they can run independently. To achieve load balancing in task node assignment, a bin-packing algorithm is used on the assumption that the relative work associated with a sub-forest of the pattern tree can be heuristically estimated. A minimum-cut bipartite graph partition model is used to reduce the degree of overlap in the local database while keeping the load balanced. A detailed receiver-initiated dynamic load-balancing strategy is used to overcome the load imbalance due to inaccurate estimation of the task run time. When a processor finishes its work, it selects a donor processor to request for work.

Zaki[10] proposed a parallel algorithm called pSPADE based on SPADE[4]. pSPADE's target machine is the SGI Origin 2000 machine which is a hardware distributed shared memory architecture. The original SPADE used a structure called the idlist to condense the sequence embedding information. The support of a k-sequence can be determined by simply intersecting the idlist's of two $(k-1)$-sequences which share the same suffix of length $(k-2)$. Sequence patterns that share the same suffix belong to the same class. SPADE recursively decomposes sequences into even smaller classes in a level-wise manner. As in other parallel PGDP algorithms, pSPADE treats the pattern space as a tree of independent suffix-based classes. In the data parallel formulation, the processors work on different partitions of the database (idlist) but synchronously process the whole pattern tree. Different idlist partition mechanisms (horizontal vs. vertical) and different synchronization granularities (single vs. level-wise) were analyzed or implemented. In the task parallel formulation, all processors have access to one copy of the database and work on different classes of the pattern tree. The static load balancing of pSPADE is similar to most other parallel PGDP algorithms, in which top level tasks are partitioned among processors using heuristic approximations. In the intra-class dynamic load balancing (CDLB) formulation, top level tasks are queued and wait for processors to fetch the tasks. However, a coarse-grained top level task may lead to inefficiencies with the CDLB. Consequently, Zaki used a sophisticated strategy called the recursive dynamic load balancing (RDLB) strategy. Whenever an idle processor is detected, a busy processor spawns nodes from its current working class and inserts them into the global task queue. Idle processors then fetch one from the task queue. Experiments show that good speedup can be achieved with the RDLB dynamic load balancing strategy.

Cong and Han[18] proposed the Par-CSP algorithm (parallel closed sequential pattern mining) which runs on distributed memory system. Their parallel algorithm is based on the state-of-the-art frequent closed sequence mining algorithm BIDE[19]. Par-CSP makes use of the depth-first-search and divide-and-conquer nature of BIDE. The tasks of mining each pseudo projection database with respect to each frequent item are distributed among processors. The task parallel paradigm here is similar to the PGDP-based parallel frequent itemset mining algorithms[12,13,15] except that Cong and Han[18] use a selective sampling technique to achieve good load balance. The technique heuristically selects small but representative portions of the projected database and runs BIDE to approximate the relative mining time for each projected database. The experimental run time is later used for static task assignments. The additional mining time posed by the sampling is relatively small. Cong and Han[20] also abstracted their sample-based framework and for use on parallelizing FP-growth[3] and PrefixSpan[8].

2.3 Parallel frequent graph mining

Transactions in a graph database are connected graphs which are usually undirected and labeled. Several FGM algorithms have been proposed in recent years including Subdue, MolFea, FSG , MoSS/MoFa, gSpan, CloseGraph, FFSM, and Gaston.

Mining frequent graphs (e.g., molecular fragments) is even more challenging than frequent sequence mining due to the inherent computational complexity of the graph-based algorithms and intrinsic graph/ subgraph isomorphism test requirements. The subgraph isomorphism problem for general graphs has been proved to be an NP-complete problem. In parallelizing FGM algorithms, the irregularity of the graph pattern lattice can lead to severe load imbalance in task partitioning. Thus, parallelizing the graph mining algorithm is nontrivial because of the need for a sophisticated load balancing strategy and a proper parallelism granularity.

The MoFa algorithm[1] was originally developed to discover connected discriminative molecular fragments for drug discovery. MoFa models the molecular fragments as attributed graphs, in which vertices represent atoms and edges represent bonds between atoms. The discrimination ability of a fragment is expressed by a dual support constraint, where an interesting fragment is frequent in an active molecular set and infrequent in another non-active set. Despite the particularity of the problem definition of MoFa, this does not harm its generality as a generic frequent subgraph discovery algorithm. MoFa conducts a depth-first search on the frequent fragment search tree and extends one bond if looks hopeful. An embedding list is used to record the exact position information of the fragment in the involved molecules, which makes the fragment grow more precisely. To solve the duplicate pattern generation problem in a fragment lattice traversal, MoFa uses a local order based structural pruning strategy, which states that the algorithm does not need to extend an atom which is inserted before the last extended atom.

Fatta and Berthold[21] proposed a distributed frequent subgraph mining algorithm based on MoFa to discover patterns in molecular compounds in grid environment. The distributed algorithm is based on a master-slave paradigm where each worker machine has local access to the entire active database. The job management machine simply prunes the search tree to generate an independent sub-task for the worker machine. Intermediate mining states, such as the order in which atoms were added, have to be included in the task assignment message. The difficulty in estimating the sub-task complexity makes dynamic load balancing desirable. The job manager keeps a job pool by requesting the worker machine to generate a task from its own running task. Whenever the worker machine is idle or has just finished its job, it gets a new task from the job pool. The load balance and excessively fine job granularity are balanced using three simple heuristic constraints based on the assumption that a search tree node can be pruned as a spawned sub-job. The constraints are that the size and support must be big enough and that the node must have enough siblings. These constraints guarantee that the worker does not starve itself while providing a nontrivial part of its search tree to others. Another heuristic is used by the job manager to decide which worker machine should first be solicited to spawn a new sub-job based on the observation that a long-running job is inclined to be further partitioned. The naive data partition of the complement dataset is not applicable for general frequent graph mining and goes beyond the focus of this paper.

Later, Fatta and Berthold[22] present another work which is similar to but more sophisticated than the earlier one[21]. The new algorithm adopts a P2P framework rather than the master-slave paradigm in which every machine is a donor and a worker at the same time. Second, its search space partitioning is enhanced with a more elegant job description using the enhanced-SMILES representation of fragments, a conditional compressed dataset transformation, and a reduction operation with a logical communication tree. When a worker has finished his job, it selects a donor among other workers to spawn a new job. Donors with longer running mining tasks have higher probabilities of being selected. This is the kernel of the so-called ranked-random polling (RRP) dynamic load balancing strategy. The ranking information is kept at a centralized machine. Finally, this distributed algorithm changes the third node pruning precondition to a more rational heuristic, that is, the order, in which the last extended atom was added, must be small enough, because nodes fulfilling this constraint tend to have more branches due to the structural pruning technique adopted in the original sequential algorithm, MoFa.

This reduces the chance of generating trivial sub-jobs.

Meinl et al.[23] implemented a parallelized version of MoFa on a shared memory machine. The globally shared data structures are the graph database (read only), the idle worker processor list (synchronized), and the global frequent fragment set (synchronized). A stack structure is used in each processor to track its depth first search path. Each worker processor also maintains its locally mined frequent fragment set which is finally merged into the global set at the proper time. At first, one processor starts mining on the whole database. Before a working processor goes beyond one search step, it checks the global idle processor list to find a co-worker and donates out part of its working stack. To escape the possible load imbalance resulting from the stack splitting, they use alternation-splitting in which every other stack element is delivered from the busy processor to an idle processor. Because the pruning strategy in MoFa cannot avoid generating duplicate frequent fragments, the algorithm must perform a graph isomorphism test and the merging of the locally mined fragments into global ones can be very time consuming. The authors gave a detailed analysis of the synchronization and efficiency of the merging process, e.g., the granularity of the locks.

Because MoFa and gSpan[9] have similar depth first search nature, Meinl et al.[24] also parallelized gSpan using a similar methodology with the so-called work stealing technique instead of work donation. In contrast to the work donation method, the work stealing technique uses a global busy processor list with idle processors actively requesting one of the busy processors for part of its work stack. Tests show that this method achieves higher speed up.

Chip multiprocessing (CMP) is a type of recently developed microprocessor (also known as multi-core systems), where multiple processor cores are placed on a single chip and share the same memory space. The design space of the algorithms specified for CMP is similar to that for shared memory system with consideration of the special CMP characteristic of lower inter processor communication cost and limited off chip bandwidth.

Buehrer et al.[25] proposed a parallelized graph mining algorithm for the CMP architecture. Their algorithm employs a depth first search on the frequent graph tree with each candidate extension from a tree node being a task unit. This guarantees fine granularity of the task which is desirable for the CMP structure. The canonical representation of gSpan and the embedding list technique of MoFa are used as enhancements. The algorithm uses a distributed task queuing model in which each processor core performs enqueue and dequeue operations on its own task queue. If a processor core's queue is empty, it searches other core's queue for work. If all queues are empty, it sleeps and waits to be wakened by another core which has a nonempty queue. To minimize off chip bandwidth, the algorithm maximizes the possibility that the graph and its induced graph are mined on the same core. A pointer-based compact embedding list is used to reduce memory consumption. Both optimization techniques actually utilize the concept of a pseudo projection database.

3 Conclusions and Future Work

The characteristics of the reviewed parallel FPM algorithms are summarized in Table 1. These algorithms are designed to solve various types of pattern discovery problems and target distinct parallel environments. Nevertheless, they share some common characteristics. Most of the recently developed parallel FPM algorithms are used to implement their parallelism based on the pattern lattice/tree traversal paradigm. The CGT pattern enumeration suffers from superfluous candidate patterns and has become unfashionable. Of the reviewed algorithms, only FIMonClumps and N/S/HPSPM fall into this class. FIMonClumps mixes existing CGT based parallel methods to run on a hybrid Clumps system. N/S/HPSPM are analogies of the CGT based parallel FIM philosophy for solving FSM problems. MLFPT, PFP-tree, and Inverted Matrix utilize the divide-and-conquer pattern enumeration scheme of the FP-growth algorithm. Many serial FSM algorithms traverse the pattern lattice in a prefix tree or suffix tree manner. Accordingly, a popular FSM task parallelism method is to partition the tree nodes of a certain level. Par-CSP and the static formulation of pSPADE partition the top level nodes, while pTPSM partitions a specified k level. The dynamic formulation of pSPADE can recursively partition the child nodes of a pattern to spawn a new task. All the surveyed parallel FGM algorithms base their parallelism on partitioning the depth first search tree. They use a variety of heuristics for load balancing with busy workers donating

Table 1 Typical parallel frequent pattern mining algorithms

Dataset type	Parallel algorithm name and Ref.	Serial prototype	Target platform	Load balancing strategy	Characteristics
Itemset	FIMonClumps*, [12]	Apriori[CGT]	Clumps	Static	Specialized for clumps
	MLFPT, [13]	FP-growth[PGDP]	SMP	Static	N/A
	PFP-tree, [13]	FP-growth[PGDP]	DMM	Static	Partially replication FP-tree
	Inverted Matrix, [15]	N/A[PGDP]	DMM	Static	Reasonable SLB
Sequence	N/S/HPSPM, [16]	GSP[CGT]	DMM	Static	Analogical from parallel FIM
	Par-CSP, [17]	BIDE[PGDP]	DMM	Static	Sample based
	pSPADE, [10]	SPADE[Hybrid]	DSM	Static, Dynamic	Suffix tree, recursive DLB strategy
	pTPSM*, [5]	TreeProjection [PGDP]	DMM	Static, Dynamic	Dedicated selective database replication
Graph	msMoFa*, [21]	MoFa[PGDP]	GRID	Dynamic	Job pool, spawning job with constraints
	p2pMoFa*, [22]	MoFa[PGDP]	GRID	Dynamic	Ranked-random polling DLB strategy
	smpMoFa*, [23]	MoFa[PGDP]	SMP	Dynamic	Pattern stack split and donation
	smpGSpan*, [24]	gSpan[PGDP]	SMP	Dynamic	Work stealing
	cmpFGM*, [25]	N/A[PGDP]	CMP	Dynamic	Distributed task queuing model

Note: The * after the names of some algorithms means that the original authors did not give an explicit name for their algorithm so it was named here.

parts of their DFS path stacks to idle workers.

Static load balancing in parallel pattern discovery has low communication overhead but requires estimates of the relative mining time for each task unit, whose precision is often not acceptable especially on skewed datasets and complex structural data (e.g., graph data). Table 1 shows that all parallel FGM algorithms and some of the FSM algorithms employ dynamic load balancing strategies rather than static strategies. Another observation is that dynamic load balancing is very desirable for discovering patterns from graph databases especially in heterogeneous environments. The precondition of utilizing a dynamic load balancing strategy is that the mining task associated with a pattern can be recursively partitioned into smaller ones. As an example, pSPADE makes use of the concept of the equivalent class for spawning subtasks needed by an idle processor. Another example is the pattern tree nodes based job scheduling in msMoFa[21].

Another vital issue in parallel FPM is database replication. The database can be partitioned among processors logically or physically with no overlaps, but because a pattern may be arbitrarily embedded in any transaction, a database exchanging process between processors is unavoidable for a processor to access all its needed transactions. Some algorithms simply duplicate the whole database on each processor to reduce the communication overhead. Examples are Par-CSP, task parallelism in pSPADE, and all the reviewed parallel FGM algorithms. However, this does not efficiently use the aggregate disk or memory space of a parallel platform. Therefore, a tradeoff is needed. The minimum-cut bipartite graph partition model in pTPSM[5] may be useful for future parallel FPM algorithm designs. Another potential solution is to design a new database representation which can inherently facilitate database partitioning and reduce local database overlapping such as the idlist partitioning based data parallel formulation in pSPADE.

Most parallel FPM algorithms are designed for traditional SMP (shared memory) or DMM (distributed memory) systems. Some algorithms have been invented for discovering patterns on hybrid systems like Clumps, heterogeneous environments like GRID, and even on new processor architecture like CMP (chip multiprocessing) and SMT (simultaneous multithreading). However, much more work is needed to parallelize sophisticated FPM algorithms on these emerging high performance computing architectures.

References

[1] Borgelt C, Berthold M R. Mining molecular fragments: Finding relevant substructures of molecules. In: Proceedings of the 2002 IEEE International Conference on Data Mining. Maebashi, Japan, 2002: 51-58.

[2] Agrawal R, Srikant R. Fast algorithms for mining association rules. In: Proceedings of the 20th VLDB Conference. Santiago, Chile, 1994: 487-499.

[3] Han J, Pei J, Yin Y. Mining frequent patterns without candidate generation. In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data. Dallas, USA, 2000: 1-12.

[4] Zaki M J. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning Journal*, 2001, **42** (1-2): 31-60.

[5] Guralnik V, Karypis G. Parallel tree-projection-based sequence mining algorithms. *Parallel Computing*, 2004, **30**(4): 443-472.

[6] Srikant R, Agrawal R. Mining sequential patterns: Generalizations and performance improvements. In: Proceedings of the Fifth International Conference on Extending Database Technology. Avignon, France, 1996.

[7] Inokuchi A, Washio T, Motoda H. An Apriori-based algorithm for mining frequent substructures from graph data. In: Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery. Lyon, France, 2000: 13-23.

[8] Pei J, Han J, Mortazavi-Asl B, et al. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In: Proceedings 2001 International Conference on Data Engineering. Heidelberg, Germany, 2001: 215.

[9] Yan X, Han J. gSpan: Graph-based substructure pattern mining. In: Proceedings of the IEEE International Conference on Data Mining. Maebashi, Japan, 2002: 721-723.

[10] Zaki M J. Parallel sequence mining on shared-memory machines. *Journal on Parallel Distributed Computing*, 2001, **61**(3): 161-189.

[11] Zaki M J. Parallel and distributed association mining: A survey. *IEEE Concurrency*, 1999, **7**(4): 14-25.

[12] Jin R, Agrawal G. An efficient association mining implementation on clusters of SMP. In: Proceedings of the 15th International Parallel and Distributed Processing Symposium. San Francisco, USA, 2001: 156.

[13] Zaïane O R, El-Hajj M, Lu P. Fast parallel association rule mining without candidacy generation. In: Proceedings of the 2001 IEEE international Conference on Data Mining. San Jose, USA, 2001: 665-668.

[14] Javed A, Khokhar A. Frequent pattern mining on message passing multiprocessor systems. *Distrib. Parallel Databases*, 2004, **16**(3): 321-334.

[15] El-Hajj M, Zaiane O R. Parallel association rule mining with minimum inter-processor communication. In: Proc. 14th International Workshop on Database and Expert Systems Applications. Prague, Czech Republic, 2003: 519.

[16] Shintani T, Kitsuregawa M. Mining algorithms for sequential patterns in parallel: Hash based approach. In: Proceedings of the Second Pacific-Asia Conference on Research and Development in Knowledge Discovery and Data Mining. Melbourne, Australia, 1998: 283-294.

[17] Agrawal R, Shafer J. Parallel mining of association rules. *IEEE Trans. Knowledge and Data Eng.*, 1996, **8**(6): 962-969.

[18] Cong S, Han J, Padua D. Parallel mining of closed sequential patterns. In: Proceeding of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining. Chicago, USA, 2005: 562-567.

[19] Wang J, Han J. BIDE: Efficient mining of frequent closed sequences. In: Proceedings of the 20th International Conference on Data Engineering. Boston, USA, 2004: 79-91.

[20] Cong S, Han J, Hoeflinger J, Padua D. A sampling-based framework for parallel data mining. In: Proceedings of the 10th ACM Symposium on Principles and Practice of Parallel Programming. Chicago, USA, 2005: 255-265.

[21] Fatta G D, Berthold M R. Distributed mining of molecular fragments. In: Proceeding of IEEE International Conference on Data Mining, Workshop on Data Mining and the Grid. Brighton, UK, 2004: 1-9.

[22] Fatta G D, Berthold M R. Dynamic load balancing for the distributed mining of molecular structures. *IEEE Transactions on Parallel and Distributed Systems*, 2006, **17**(8): 773-785.

[23] Meinl T, Fischer I, Philippsen M. Parallel mining for frequent fragments on a shared-memory multiprocessor. In: LWA 2005. German Research Center for Artificial Intelligence, 2005: 196-201.

[24] Meinl T, Worlein M, Fischer I, et al. Mining molecular datasets on symmetric multiprocessor systems. In: Proceedings of the 2006 IEEE International Conference on Systems, Man and Cybernetics. Taipei, China, 2006: 1269-1274.

[25] Buehrer G, Parthasarathy S, Kim D, et al. Towards data mining on emerging architectures. In: Proceedings of 9th SIAM Workshop on High Performance and Distributed Mining. Bethesda, USA, 2006.