Disk-based pattern mining

Data Base and Data Mining Group of Politecnico di Torino

Elena Baralis, Tania Cerquitelli

Politecnico di Torino

Turin, May 13th 2009

Main-memory frequent pattern mining

- Continuous improvement with respect to baseline techniques for efficient computation of frequent itemsets
  - Item-covers [Zak00]
  - COFI-Tree [El-04]
  - Patricia-Trie [Pie03]
  - Prefix-Trie: Prefix-Tree [Gra03], AFOPT [LiL03]
  - Array-based: LCM v.2 [Uno04]
  - Hybrid structures: LCM v.3 [Uno05], CGAT [Bas06]
- Silver bullet not yet available
  - every technique works better with a given data distribution

Incremental mining of frequent patterns

- Many real-life databases are updated by periodically incoming business information
  - E.g., data evolve over time
- Most techniques require frequent itemset recomputation
- Some algorithms incrementally update ad-hoc data structures to simplify extraction
  - CATS-Tree [Che03]
  - CanTree [Leu05]
  - FP-Tree based structure [Adn06]
  - INUP_Tree [HeZ07]
  - I-Forest [Bar08]

Large scale frequent pattern mining

- Technological advances allow gathering an increasingly large amount of data
  - e.g., in the science, engineering, and business areas
  - database size > 100 GB
- Our ability to collect data far outstrips our capability to analyze it efficiently
  - advanced strategies to speed up and scale up data mining algorithms are needed

Large scale frequent pattern mining

- Main-memory data mining algorithms
  - exploit ad-hoc main-memory data structures to efficiently extract knowledge
    - memory resident data structures to represent the original dataset in main-memory
    - rely on the available physical memory
    - may run out of memory when the analysis is performed on very large databases [Goe04,Vaa04]
  - are computationally complex
- To overcome the main memory size bottleneck
  - *disk-based* frequent pattern mining algorithms

DBMG

Disk-based frequent pattern mining

- The analysis is split in two steps
  - Given the original dataset, a persistent (possibly lossy) representation of its data is stored in secondary memory
    - clever and compact data structures are needed
  - Itemset extraction is performed on relevant portions of these data structures
    - only a reduced portion of data is loaded into main memory to be processed by the current mining process

DBMG

Proposed approaches

- Disk-based approaches
  - B+tree-based indices [Ram02]
  - Inverted Matrix [El03]
  - Diskmine [Gra04]
  - TDD and ST-Merge Method: suffix-tree [Tat04,Tia05]
  - I/O conscious optimizations [Bue06]
  - TRELLIS: suffix tree indexing [Pho07]
  - DRFP-tree [Adb09]
- Tight integration of pattern extraction in a relational DBMS
  - IMine index integrated into PostgreSQL [Bar05,Bar09]

B+tree-based indices

- Proposed by Ramesh et al. in [Ram02]
  - For a vertical data representation (ECLAT-Based [Zak00])
    - for each item, the list of transactions (tidlist) containing the item is stored
    - uses tidlist intersections to compute the support of an itemset
    - coarse grained index: Itemset ID is the key and the tidlist is a variable length data field
    - fine grained index: (Itemset ID, tid) is the key and no data field is associated with the key
  - For a horizontal data representation (APRIORI-Based [Agr94])
    - transactions are stored as (tid, itemset)
    - coarse grained index: tid is the key and the itemset is a variable length data field
    - fine grained index: (tid,item) is the key and no data field is associated with the key
- Drawback
  - performance is usually worse than, or at best comparable to, flat file mining

Inverted Matrix

- Proposed by ElHajj and Zaiane in [El03]
  - A disk-based data structure is exploited to store the original dataset
    - inverted matrix layout
      - each item is associated with all transactions in which it occurs (i.e., an inverted index)
      - each transaction is associated with items using pointers
  - The COFI-Tree (Co-Occurrence Frequent Item Tree) main-memory data structure
    - similar to the conditional FP-Tree
    - used to generate the frequent itemsets
- Drawback
  - It is specifically suited for very sparse datasets, characterized by a significant number of items with unitary support

Diskmine

- Proposed by Grahne and Zhu in [Gra04]
  - Large databases are materialized on disk in different projected databases whose size fits in main memory
    - recursive projections to partition the data until it fits in main memory
  - The in-memory FPgrowth algorithm is exploited to mine the projected data sets
    - the complete set of frequent itemsets is computed by taking the union of the itemsets mined from each projection
- Drawbacks
  - It requires several (costly) accesses to the potentially large number of projected datasets
  - It may need significant disk space to store projections

I/O conscious optimizations

- Proposed by Buehrer et al. in [Bue06]
  - A slight variation of the FP-Tree data structure to compactly represent original database on disk
    - each node stores item identifier, local support, and node father pointer
    - node link pointers and global support are stored in a separate structure
  - Approximate hash sorting techniques to minimize the number of page faults during the prefix-tree construction
    - frequent transactions are redistributed into a partition of blocks and approximately sorted
    - each block is implemented as a separate file on disk
    - the global prefix-tree is built by processing the files in order

I/O conscious optimizations

  - Improving spatial data locality
    - objective: reducing the number of reads when accessing the prefix-tree in a bottom-up fashion
    - the global prefix-tree is reallocated in virtual memory to obtain the tree in depth-first order
  - Improving temporal data locality
    - objective: maximizing reuse of the prefix tree once it is fetched into main memory
    - the tree is broken down into fixed size blocks of memory (page blocks) along paths of the tree from the leaf nodes to the root
    - blocks may be partially overlapped
- Drawbacks
  - Data locality requirements are different for different data structures and mining algorithms
  - Different I/O conscious techniques should be devised for different mining approaches

IMine

- Proposed by Baralis et al. in [Bar09]
  - Index integrated into PostgreSQL
    - The index provides a complete representation of the original database
      - a prefix-tree, stored in a relational table, encodes in a unique structure the complete dataset
        - no support threshold enforced
      - each node of the tree contains supplementary information to support more flexible data access methods
      - a B+Tree structure provides selective access to the prefix-tree disk blocks during the extraction process
  - Data access functions
    - support the enforcement of various constraint categories (e.g., support constraint, item constraint)
    - support different extraction approaches
      - projection-based algorithms (e.g., FP-growth [Han00])
      - level-based algorithms (e.g., APRIORI [Agr94])
      - array-based algorithms (e.g., LCM v.2 [Uno04])

IMine

- I/O optimization strategies
  - Correlation analysis is performed to discover data accessed together
  - Correlated information is stored in the same block to minimize the number of physical data blocks read during the mining process
- Frequent pattern extraction
  - Available implementations for
    - FPGrowth [Han00]
    - LCM v.2 [Uno04]
  - Algorithms characterized by
    - different in-memory data representations (e.g., array list, prefix-tree)
    - different techniques for visiting the search space
- Drawbacks
  - Can not deal with dataset size > 50 GB
  - Depends on PostgreSQL internals evolution

Query languages

- Complex extraction requests cannot be specified directly in the mining process
  - Constraints on support&confidence (easy!)
  - Constraints on rule structure
  - Constraints on correlation structure
- Proposed approaches
  - Data mining query languages
    - DMQL [Han96]
    - MINE RULE [Meo96]
    - RULE-QL [Tuz02]
- Drawbacks
  - Focus on language expressiveness, not on performance
  - No implementations in real systems

Querying frequent patterns

- Discovered knowledge needs to be efficiently stored and accessed
  - postprocessing of large mining results
- Proposed approaches
  - Disk-based structures to efficiently store mined knowledge
    - Group bitmap index [Mor98]
    - CFP-Tree (Condensed Frequent Pattern-Tree) [Liu03,Liu07]
- Drawbacks
  - Size of extracted rule set (also with compact forms) is larger than original dataset
  - Proposed approaches are not able to deal with large scale results (e.g., GB of data)

Open research issue

- Proposed algorithms do not scale well when applied to current very large databases
  - DB size > 100 GB
- Dealing with disk-resident data
  - is the most promising option
    - apart from special hardware solutions (e.g., parallel systems)
  - affects performance
    - retrieving data from disk is significantly slower than accessing data in RAM
  - requires ad-hoc approaches
    - techniques for cache and buffer management
- Should leverage on techniques for database system indexing

Future directions

- Study and design novel *hybrid* disk-based data representations
  - to compactly store huge amounts of data on secondary memory
    - for any data distribution (e.g., dense, sparse)
    - for varying data distributions over the same dataset
  - examples: tree-based structure, array-based structure, hash table, bitmap indices
- Clever physical data representations
  - to reduce the number (and cost!) of disk reads
  - to limit the amount of memory used in the mining process
  - should exploit *data locality*

Future directions

- Study and design novel data retrieval algorithms to directly manage I/O
  - objective: selectively loading in main memory only the projection of the original database useful for the current mining process
    - exploit clever physical representations
  - disk-resident memory can be directly managed by the programmer through the file system
    - hard to program, system dependent
- Exploit “best” frequent pattern algorithm for the dataset at hand
  - analyze data distribution
  - define selection options

DBMG

Mining structural patterns

- Graph databases
  - Databases with data modeled as graphs
    - e.g., XML documents, web logs, citation networks, chemical structures
- Graph pattern mining, with and without constraints, means to find the common substructures from a collection of graphs
  - Graph databases are very large and sometimes cannot be mined in main memory

DBMG

Mining structural patterns

- Different ad-hoc data structures have been devised to efficiently perform the frequent graph pattern mining from graph databases
  - Main-memory approach
    - AGM (Apriori-like algorithm) [Ino00]
    - FSM [Kur01]
    - SUBDUE (approximate algorithm) [Kur01]
    - gSpan (depth-first approach) [Yan02]
  - Disk-based approach
    - GraphMiner (Index support for gSpan algorithm) [Wan04, Wan05]
- Frequent tree pattern mining
  - Main-memory approach
    - FREQT [Asa02]
    - TREEMINER [Zak02]
- Drawback
  - Proposed approaches are not able to deal with terabytes of data