VERTICAL MINING

E0 261

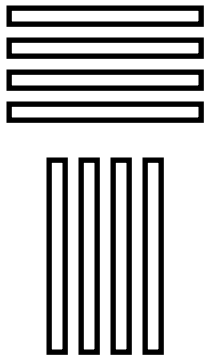
Jayant Haritsa

Computer Science and Automation

Indian Institute of Science

Data Layouts for Mining

- Table Organization
  - Horizontal (transactions / rows)
  - Vertical (items / columns)

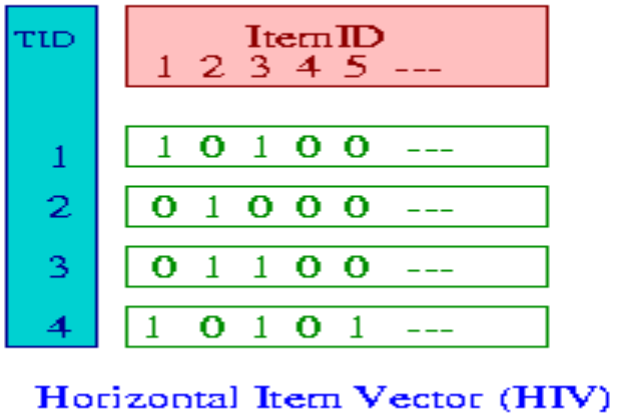


- Data Representation
  - Value List (only presence)
  - Bit Vector (presence and absence)

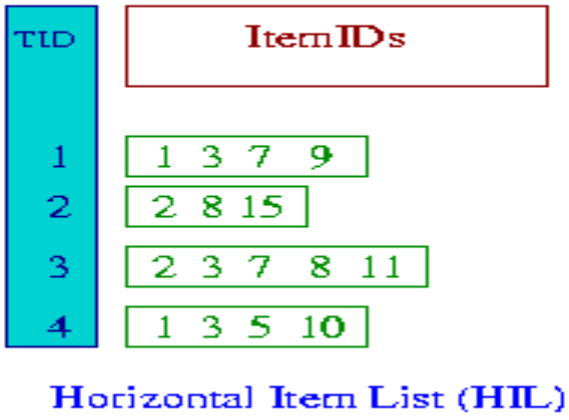
1 → 2 → 5

1 1 0 0 1

Data Layout Combinations



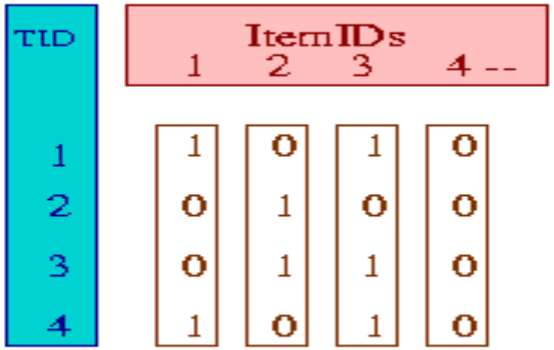
Horizontal Item Vector (HIV)



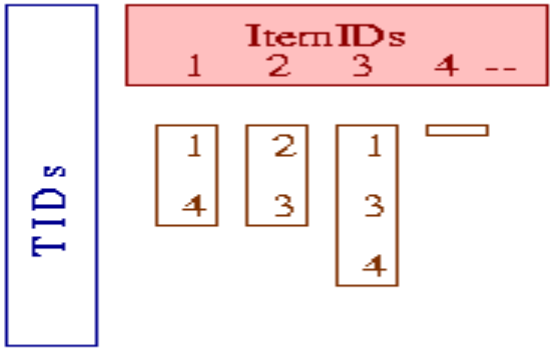
Horizontal Item List (HIL)

←Apriori

Our Approach →



Vertical Tid Vector (VTV)



Vertical Tid List (VTL)

Why Vertical Organization?

- "Natural" for association rule mining's goal of discovering correlated items (columns)
- Support counting simple (set intersections)
- No excess baggage from disk (automatic and immediate reduction of database after each scan)
- Ideal for parallel implementations (asynchronous, not level-wise, computation)
  - counting of AB can start before item C has been fully counted

Why Bit-Vector Representation?

- Tremendous scope for compression, especially since databases are typically sparse
- Vertical orientation offers better compression than horizontal since column lengths are proportional to size of database whereas row lengths are proportional to size of schema
- In fact, compressed VTV occupies much less space than HIL

Contributions

- Compressed VTV data layout — “Snake”
- VIPER snake mining algorithm
  - (Vertical Itemset Partitioning for Efficient Rule-extraction)
  - several optimizations for snake generation, intersection, counting and storage
  - “general-purpose’’ (prior vertical algorithms have restrictions on DB size, shape, contents, mining process)
  - substantial performance improvement (response time, disk space, disk traffic)
  - in some cases, beats “optimal” horizontal !

Snake-charmers

IIT-Bombay

- Pradeep Shenoy [UW Seattle, ML Manager Microsoft India]
- Gaurav Bhalotia [UC Berkeley, VP Flipkart]
- Mayank Bawa [Stanford, CEO WorkSpan]
- Devavrat Shah [Stanford, MIT professor]

- S. Sudarshan [IIT-B professor]

VIPER Mining Algorithm

First Pass of VIPER

- Snakes are created from original HIL database
- Compression using "Skinning" algorithm (based on Golomb encoding)
- F_1 is computed

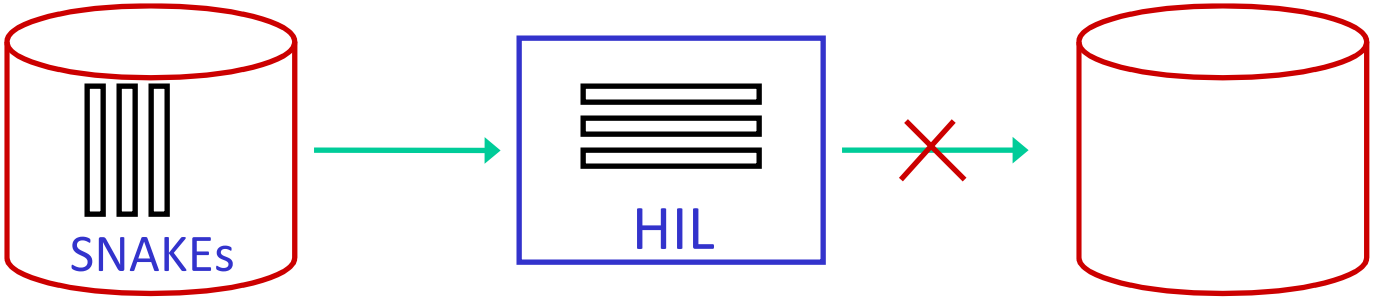
Encoding

- Run-length encoding
  - If all 1's occur in isolated manner, the RLE vector will output two words for each occurrence of a 1 – one word for preceding 0 run and one for the 1 itself. This will result in doubling size of original HIL database (where 0's are not represented). Therefore, *expansion*, not compression!

- Golomb Encoding: (f_1 f_2 … 0_{fs} f_{cnt})
  - 30-bit vector
    - 1 000 1 0000 0000 0 1 0000 0000 0 1 0000 0
  - 25-bit compressed version ($W_0 = 4$; $W_1 = 1$)
    - 10 011 10 11001 10 11001 10 1 001

Second Pass of VIPER

- Computing F_2 by intersecting all snake pairs is very expensive
- Therefore, streaming HIL database created in memory; all pairs of items in each tuple are enumerated and counted using a 2-D array
- F_2 identified, but NO snakes written to disk



Snake Writing Philosophy

- Writing of all candidate snakes to disk is very expensive and redundant since many may turn out to be infrequent
- Therefore, introduce *lag* of one pass in writing snakes to disk
- Means that only (subset of) frequent snakes are written to disk
- The “unwritten” snakes required as inputs to current pass are *dynamically* regenerated using snakes written to disk during the *previous* pass

Subsequent Passes of VIPER

- Candidate Generation (FORC algorithm)
  - (Fully Organized Candidate-Generation)
  - Generates candidates from levels *i+1* to *2i*
    (e.g. For third pass, candidates of length 3 and 4)
  - Based on equivalence class clustering technique [KDD 97]
  - Simultaneous subset detection of multiple candidates (significantly lowered computational cost)
  - Applicable to *horizontal* mining also, important advantages over AprioriGen

... Contd. ...

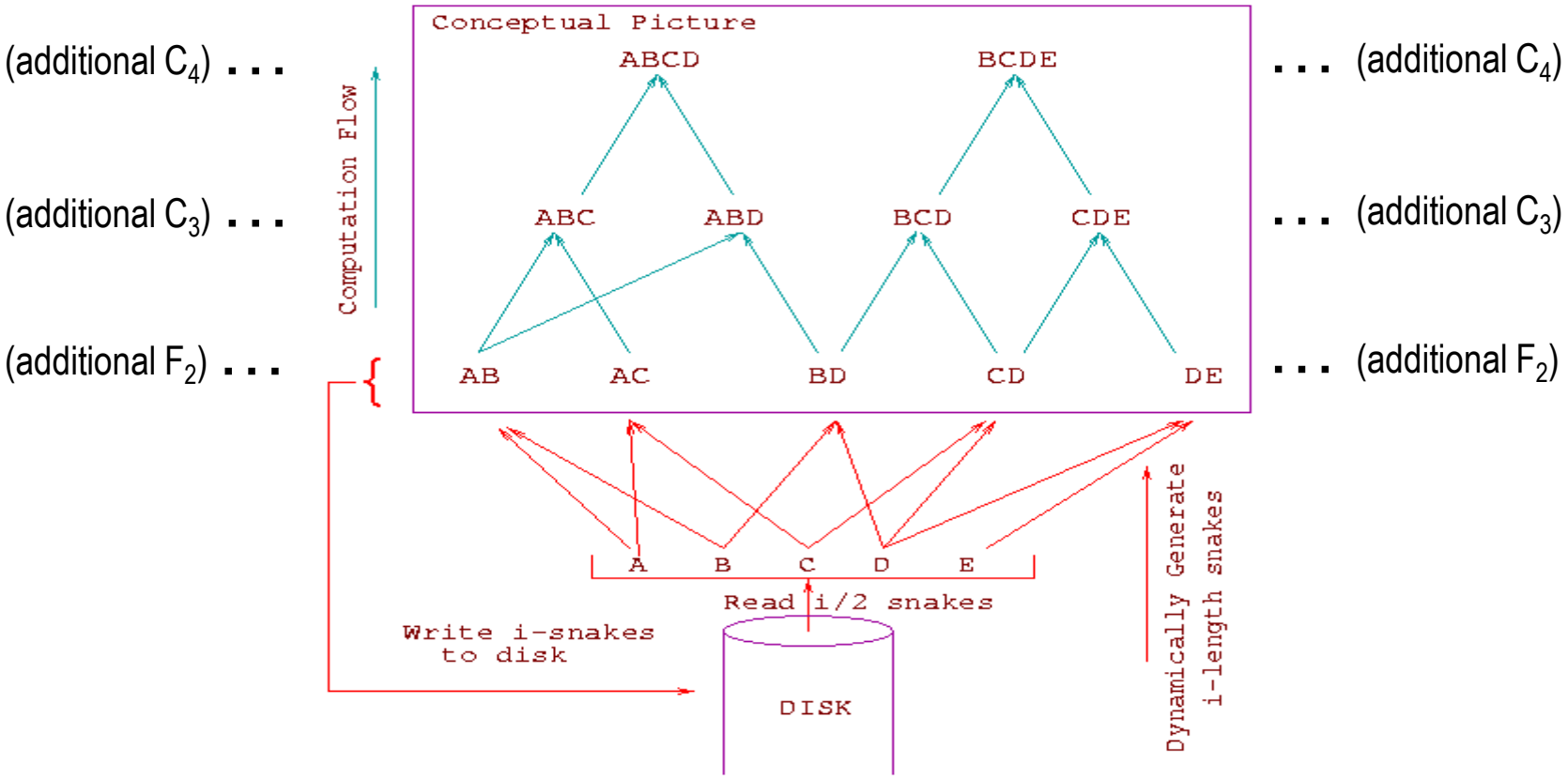
- Snake Intersection (FANGS algorithm)
  - (Fast ANding Graph for Snakes)
  - inserts candidates in DAG counting structure
  - identifies set of frequent snakes to be read in order to count these candidates
  - identifies set of frequent snakes to be written to disk for supporting counting during next pass
  - scan over database: reading, counting, and writing
  - intersection by converting snakes into streaming VTL format in memory and merge-joining these lists

Summary of VIPER Components

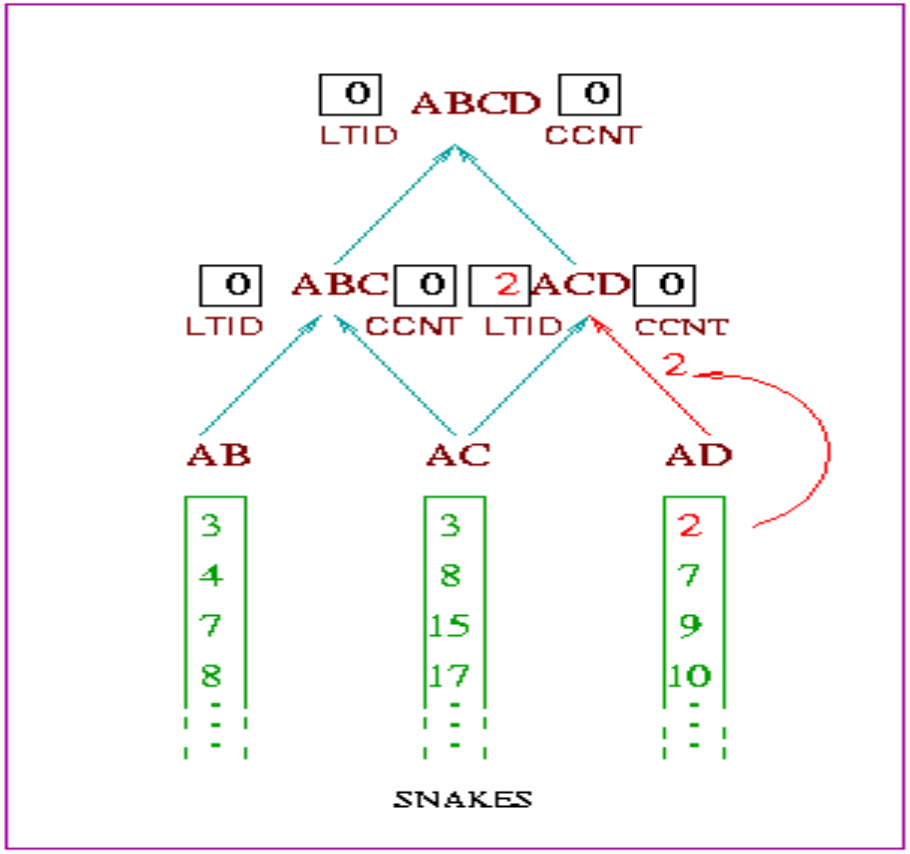
- Skinning for Snake Compression
- FORC for Snake Candidate Generation
- FANGS for Snake Intersection and Counting

FANGS Counting Algorithm

FANGS Candidate Snakes DAG (Pass 3)



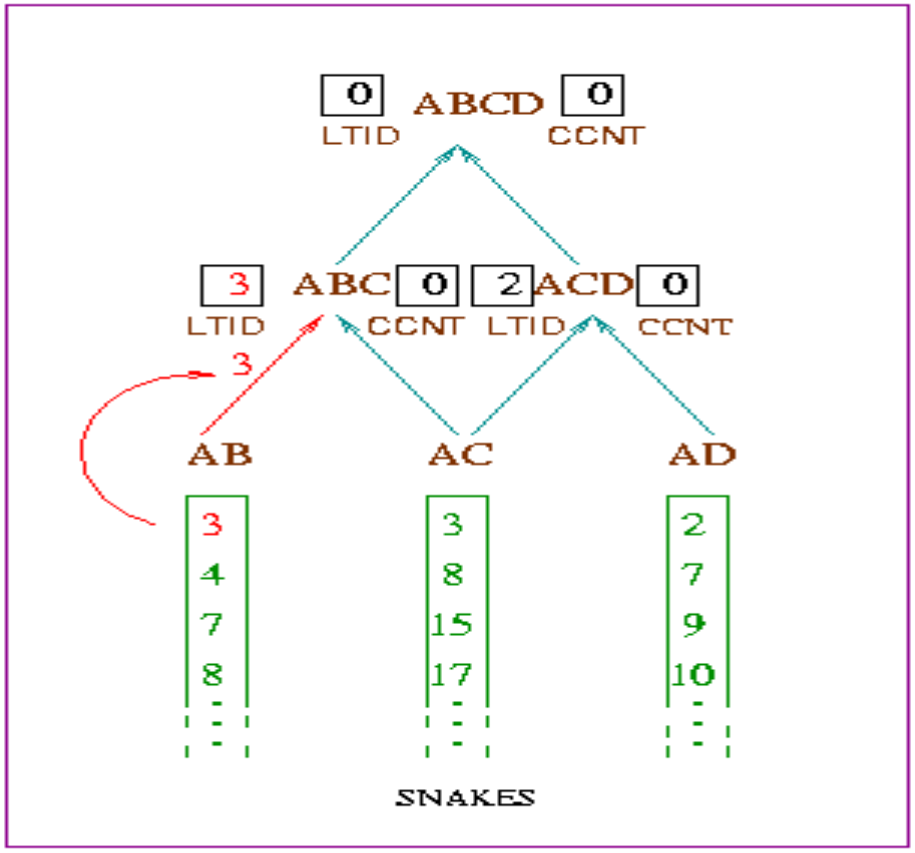
FANGS Counting Process Example



LTID : Latest TID

CCNT: Current Count

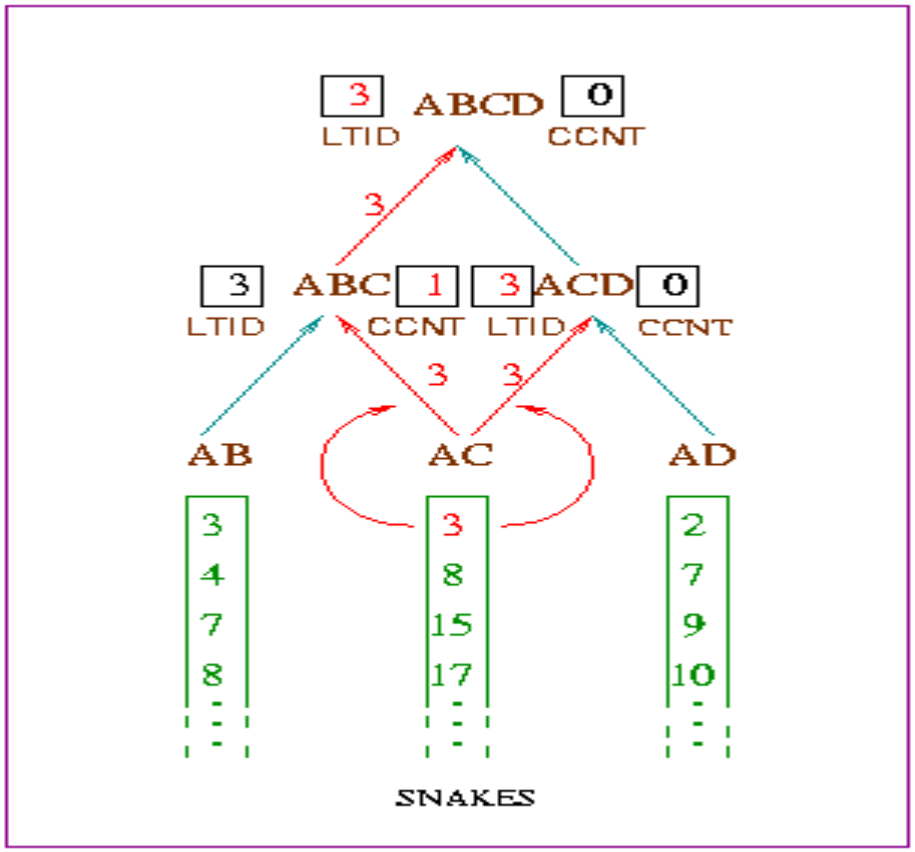
Counting Example (contd.)



LTID : Latest TID

CCNT: Current Count

Counting Example (contd.)



LTID : Latest TID

CCNT: Current Count

Optimizations in FANGS

Snake Write List and Read List Selection

- Simple solution:
  - during the pass, write out *all* the frequent i–snakes to disk (i.e., Write List = F_i)
  - after the pass, for each 2i–candidate that turns out to be frequent, choose any pair of i–snakes whose union gives the candidate as the "generator cover" to be included in the Read List

List Selection Optimizations

- Early Cover Assignment
  - Associate a pair of frequent i–snakes to each 2i–candidate *before* the pass
  - Include only these i–snakes in WriteList
- Choice of Cover
  - If multiple covers exist (e.g. (AB, CD) and (AC, BD) are both covers for ABCD), choose the pair that has maximum overlap with snakes already in the WriteList

List Selection Optimizations (contd.)

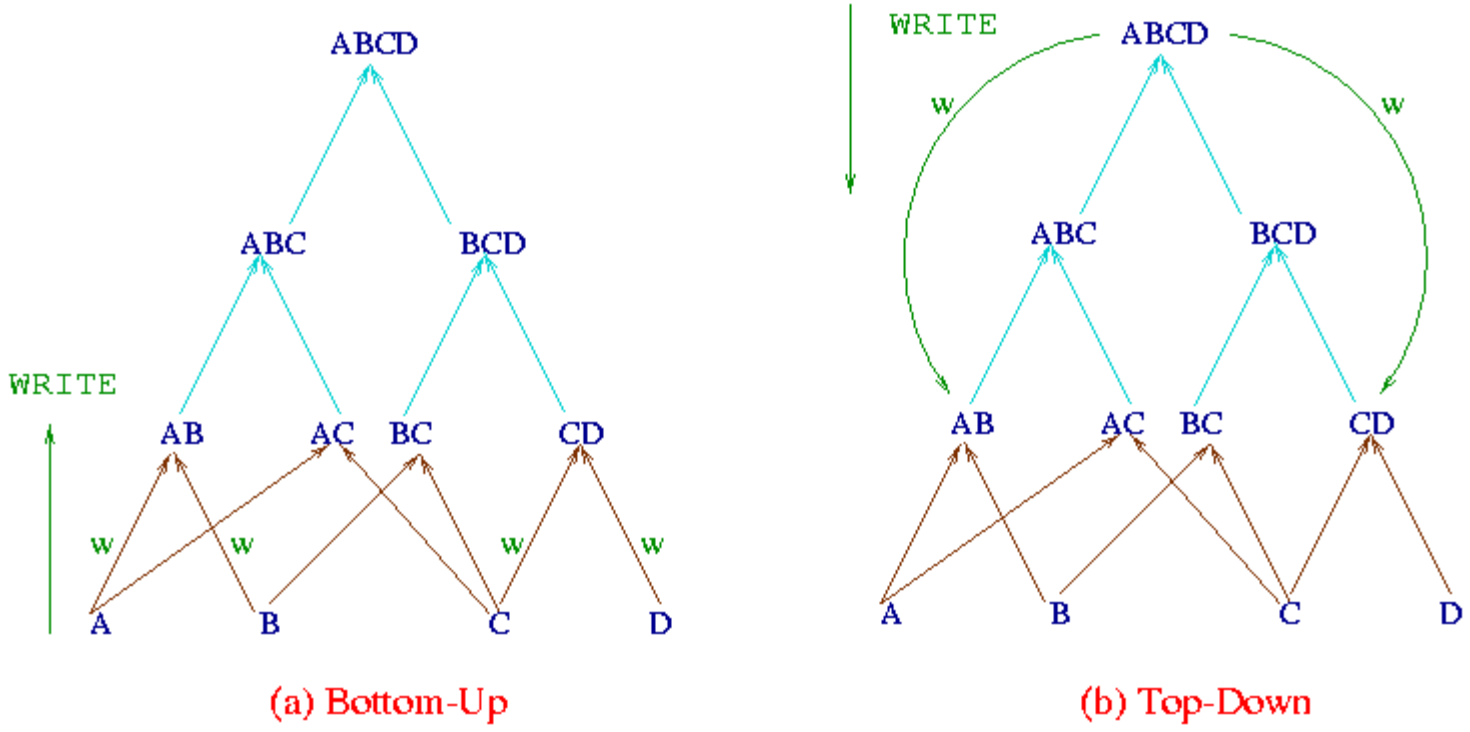
- Cover Assignment Order
  - process candidates in decreasing order of estimated support since higher support candidates will tend to have covers that overlap a larger fraction of candidates
  - for each candidate, give preference to covers comprised of itemsets with higher support, since such itemsets will be common to a larger fraction of the candidates

List Selection Optimizations (contd.)

- Alternative Cover Assignment Order
  - Give preference to *low support* snakes
  - Such snakes will be more compressed, resulting in less computational effort and disk traffic
- Choice between “a small cover of high-frequency snakes” and “a larger cover of low-frequency snakes”
  - former approach wins in our experiments

“Snake Trimming” Optimization

Increase the sparseness of snakes written to disk by top-down writes, resulting in higher compression



Performance Evaluation

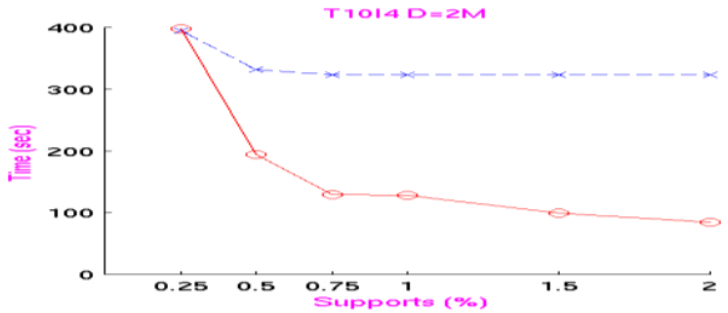
Algorithms

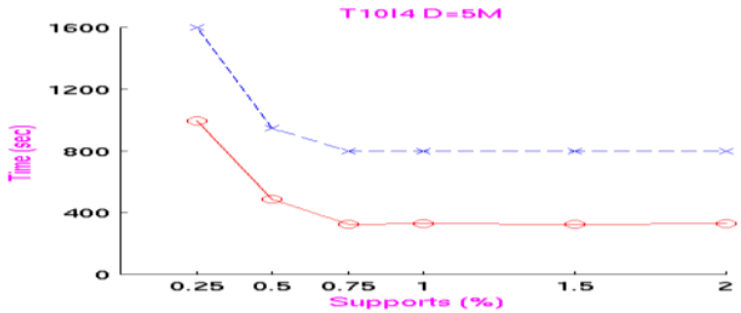
- Vertical Mining :
  - VIPER
  - MaxClique [KDD 97]
- Horizontal Mining :
  - Apriori [VLDB 94]
  - ORACLE
    - “knows” in advance identities of all frequent itemsets
    - makes single database scan to count their supports
    - uses same data and storage structures as Apriori

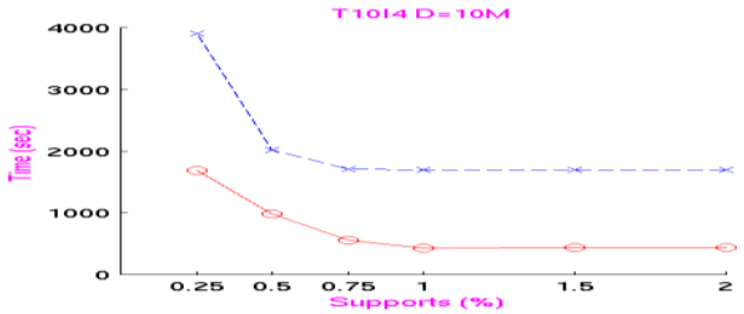
Databases

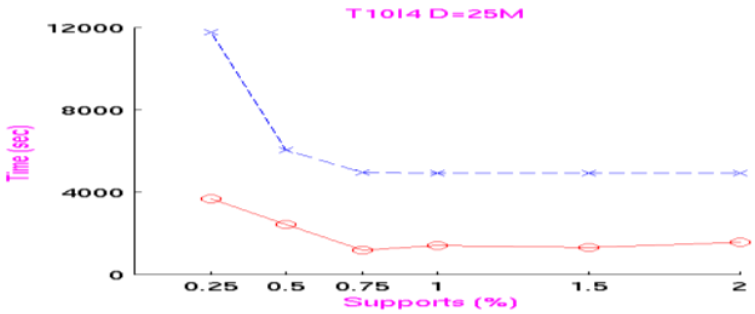
- Synthetic databases (IBM Almaden simulator)
- Includes database sizes that are significantly larger than main memory
  - 2 million tuples to 25 million tuples
- Includes “short and wide” databases [ICDE 99]
  - number of columns >> number of rows
    (e.g. On Web, large set of keywords, few documents)

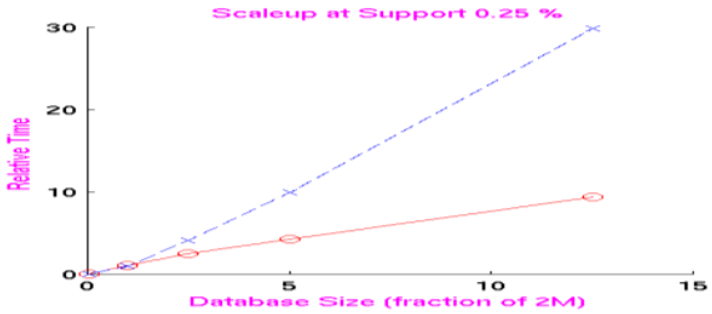
VIPER versus MaxClique

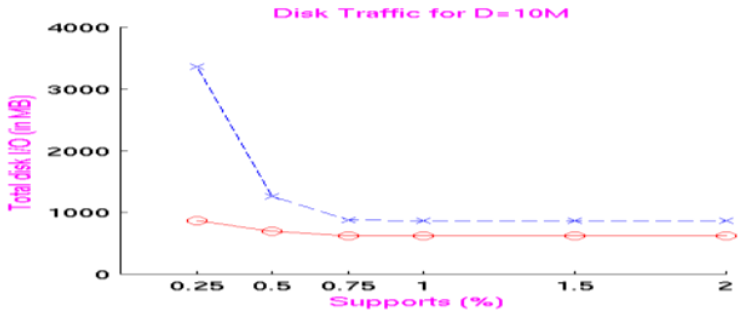








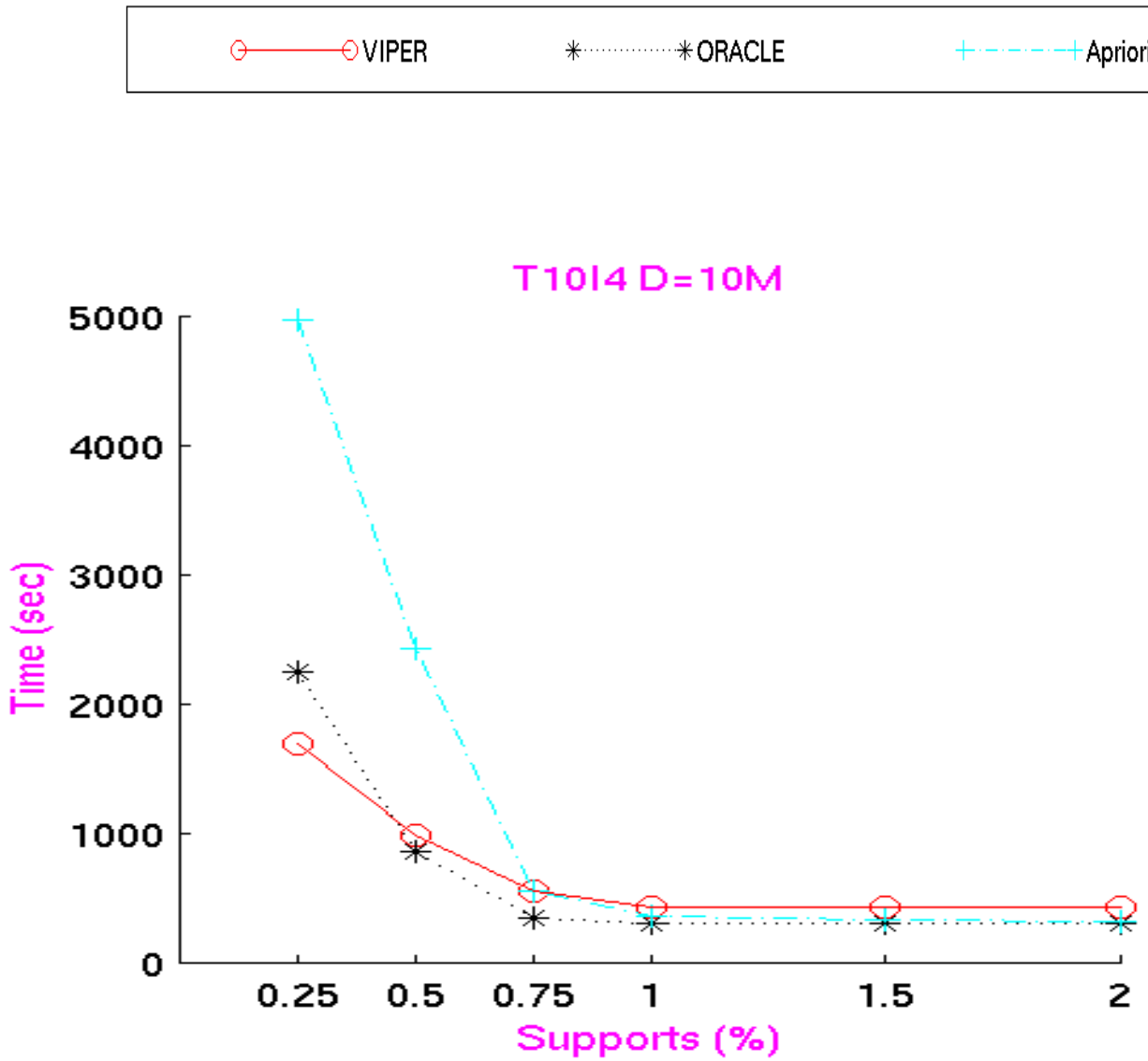




VIPER versus MaxClique (contd.)

- VIPER consistently performs better than MaxClique
- VIPER scales with database size
- VIPER's disk traffic is less than that of MaxClique
  - MaxClique reads in sameTID-list multiple times, whereas VIPER has a single scan per snake coupled with lazy snake writes

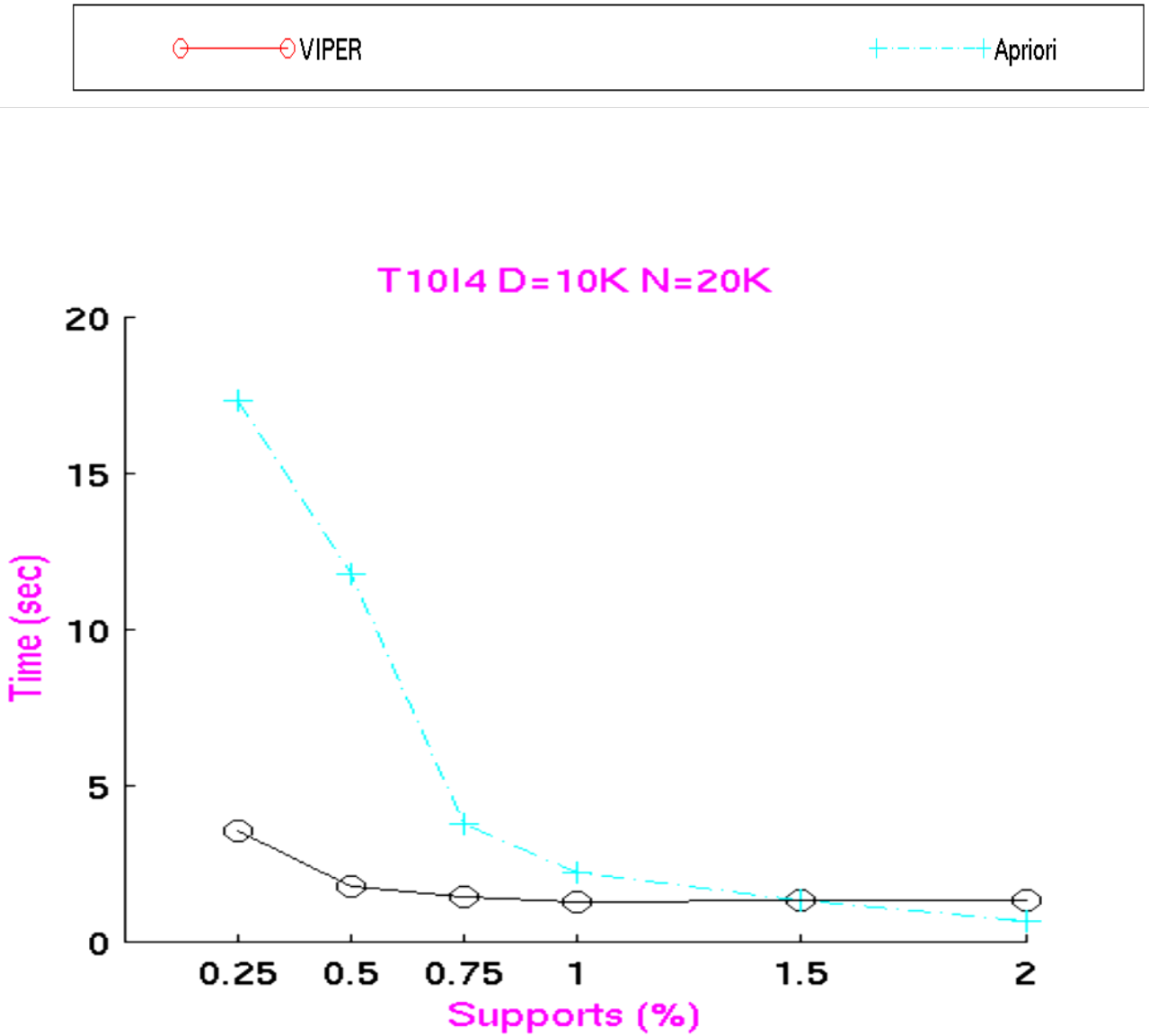
VIPER versus Apriori / ORACLE



VIPER vs Apriori/ORACLE (contd.)

- Apriori worse than VIPER at low supports because of having to make several scans
- Apriori better than VIPER at high supports is artifact of experimental setup — conversion cost from HIL to Snake predominates
- Apriori beats MaxClique over much of loading range — pre-processing times are included in execution time computations
- VIPER beats ORACLE when hash-tree processing overhead is high

Short and Wide Database



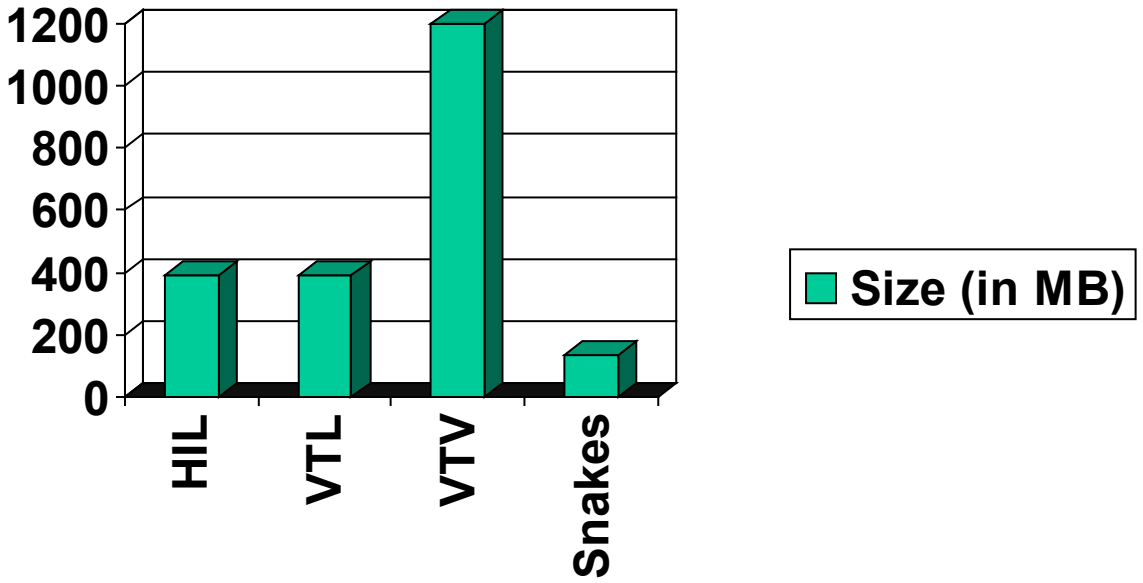
Short and Wide Database (contd.)

- Column-Wise [ICDE99] applies only to short-and-wide databases

- VIPER applies equally to both short-and-wide databases, as well as the more traditional tall-and-thin databases

Compression Statistics

T10I4 D=10M



Snake ~ 1/3 of HIL ~ 1/10 of VTV

Pruning Statistics

Database: T10I4 D=10M, support: 0.25

Starting level = 2

Candidates at level 3: 3458

Candidates at level 4: 2402

No. of 2-snakes generated: 2504

(dynamically generated to count C_3 and C_4 in current pass)

No. of 2-snakes written: 1474

(disk-written to generate relevant F_4 in next pass)

Conclusions

- VIPER aggressively materializes benefits offered by vertical layout
- Uses VTV format on disk, HIL and VTL in memory
- General-purpose and provides scaleable performance, unlike prior vertical algorithms
- Performance improvement in multiple aspects (response time, disk space, disk traffic)
- Capable of beating “optimal” version of Apriori

END VERTICAL MINING

E0 261