



A Fast Algorithm for Data Mining

Aarathi Raghu

Advisor:

Dr. Chris Pollett

Committee members:

Dr. Mark Stamp, Dr. T.Y.Lin



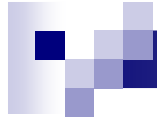
Our Work

- Interested in finding closed frequent itemsets in large databases
 - Large body of work (papers published for over 15 years)
- In this thesis, build upon the work of Lin et. al.
 - “Using Attribute Value Lattice to Find Closed Frequent Itemsets”
- In particular, our contributions include ...



Contributions

- Identify correctness issues with the algorithm and re-wrote it for clarity
- Implement the algorithm and do a performance study
- Identify implementation issues and propose solutions
- Mechanisms for improving run-time for certain data-sets – implemented a data checker and a data binner



Talk Outline

- Background
- Algorithm
- Performance Evaluation
- Conclusions



Background

- We want to efficiently mine a large database for frequently occurring patterns
 - Example: Find the set of authors whose books are frequently bought
- Prerequisites for an algorithm
 - Computational efficiency
 - Minimize number of database scans



Definitions

- Itemset (N): A set consisting of items from the database . i.e., $N \subseteq \mathcal{I}$
- Support: Number of transactions in which an itemset occurs as a subset.
- Minimum support: User specified level of support . (minsup)
- Frequent itemsets: Itemsets that satisfy minsup.



Identifying Frequent Itemsets

- Apriori (Agarwal and Srikant, 1994)
 - Commonly cited algorithm in the literature
- Approach:
 - Scan the database to identify frequent itemsets of length 1
 - From the frequent itemsets of length N , generate candidates of size $N+1$ and test
 - Stop when no new candidates are generated



Apriori

- The candidate-generation step governs the computational efficiency
- Key insight: If itemset I is **not** frequent, then any candidate that contains I is guaranteed to be **not** frequent
 - Known as *downward closure lemma*
- Use the downward closure lemma to prune the candidate-set



Improving Upon Apriori

- Mine **closed** frequent itemsets.
 - A frequent itemset N is said to be **closed** if and only if there does not exist another frequent itemset of which N is a subset.
- If F denotes the set of frequent itemsets and C denotes the set of closed frequent itemsets, then $C \subseteq F$.
- Generally, $|C| \ll |F|$ (Zaki2002)



Mining Closed Frequent Itemsets

- Several algorithms developed recently:
 - Charm, Pascal, Mafia,
- We study the algorithm of Lin et. al.



Talk Outline

- Background
- Algorithm
- Performance Evaluation
- Conclusions



Algorithm Overview

- Two phases for identifying closed frequent itemsets:
 - Phase 1: Construct a lattice of frequent itemsets
 - Phase 2: Mine the lattice in a **bottom-up breadth-first** manner to identify closed frequent itemsets



Preliminaries: Data Representation

- Data can be represented as:
 - Horizontal view : represent each row with a unique transaction identifier and a bitmap to represent items in the transaction.
 - Vertical view: represent each column with a unique identifier and a bitmap to represent transaction in which that item is involved.



Preliminaries: Data Rep. (II)

- Vertical representation allows operating only frequent itemsets.
- Bitmaps of non-frequent itemsets can be discarded leading to reduced memory footprint.
- Vertical representation of an item in the database is known as a “granule”.



Putting It All Together

- Represent lattice as a directed acyclic graph (Phase one of the algorithm)
- Augment the dag with nodes from the dag itself (Phase two of the algorithm)
- Construct closed frequent itemsets using paths in the graph
 - Start from node with in-degree 0
 - Traverse a path in the graph until we reach a node with out-degree 0



Phase One

Phase One()

1. Construct the bitmap $B(I)$ for each frequent itemset (I) in the database.
2. Set level number L of each I as 1
3. Construct the set of nodes, N , that contains I , L and $B(I)$ where $B(I) > \text{minSup}$
4. Sort the nodes based on level and bitcount . Have these in a priority queue where the priority is set as $(2^L)*B(I)$.



Phase One

5. For each node I_i in Nodes

5.1 For each sibling I_j after I_i in Nodes

5.1.1 $I = I_i \cup I_j$ and $B_{\text{comb}} = B(I_i) \cap B(I_j)$

5.1.2 If $B_{\text{comb}} > \text{minSup}$

5.1.2.1 If $B(I_i) = B(I_j)$

5.1.2.1.1 Remove I_j from Nodes

5.1.2.1.2 Replace all I_j with I (i.e $I_i \cup I_j$)

5.1.2.2 Else, if $B(I_i) \subset B(I_j)$

5.1.2.2.1 Create an edge from I_i to I_j

5.1.2.2.2 $L_j = \text{Max}(L_j, L_i + 1)$

5.1.2.3 Else, if $B(I_j) \subset B(I_i)$



Phase Two

Procedure ExpandFreqItemSet(Nodes, minsup)

1. For every node $I_i \in \text{Nodes}$

1.1 NewNodes = \emptyset , $I = I_i$

1.2 For each sibling I_j after I_i in Nodes

1.2.1 $I = I_i \cup I_j$ and $B_{\text{comb}} = B(I_i) \cap B(I_j)$

1.2.2 If $B_{\text{comb}} > \text{minSup}$

1.2.2.1 Add $I \times B_{\text{comb}}$ to the NewNode

1.2.3 Else add I_i 's parents to the NewNode

1.3 $F = F \cup I$

2. If NewNodes $\neq \emptyset$, then ExpandFreqItemSet(NewNodes, minsup)



Apriori Characteristics

■ Advantages:

- Works well for sparse databases
 - Sparse => Few frequent itemsets
- Ease of implementation

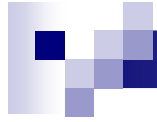
■ Disadvantages:

- Pruning efficiency is proportional to the number of frequently occurring itemsets:
 - S is the set of frequent itemsets.
 - The number of subsets of length k can potentially grow exponentially in the size of S .
- Number of database scans depends on the length of the longest frequent itemset



Overall Procedure

- Main()
- $C = \{ \}$ // set of closed frequent itemsets
- $F = \{ \}$ // set of frequent itemsets
- Construct attribute value lattice (i.e., Phase1)
- Expand frequent itemsets (i.e., Phase 2)
- For every node $I_i \in F$, add the ancestor set of I_i to C



Talk Outline

- Background
- Algorithm
- Performance Evaluation
- Conclusions



Experimental Setting

- Implemented the algorithm in Java
- Performed a set of experiments using synthetically generated data
 - Model the occurrence of an item in a transaction using a mathematical distribution
 - Distributions studied in our work:
 - Normal, Exponential, Zipf

Types of Data sets

Normal distribution

$$F(x; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/(2\sigma^2)} \quad [\text{Wikipedia}]$$

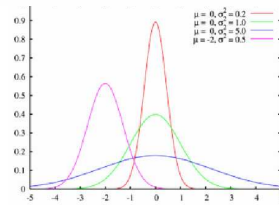
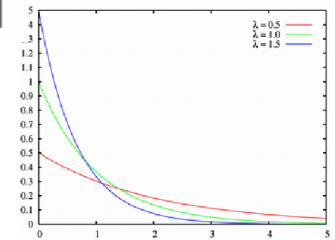


Figure 9 Normal Distribution [Wikipedia]

Exponential distribution

$$F(x; \lambda) = \lambda e^{-\lambda x} \quad [\text{Wikipedia}]$$



Zipf Distribution

$$F(k; s, N) = (1 / k^s) / (\sum_{n=1}^N 1 / n^s) \quad [\text{Wikipedia}]$$

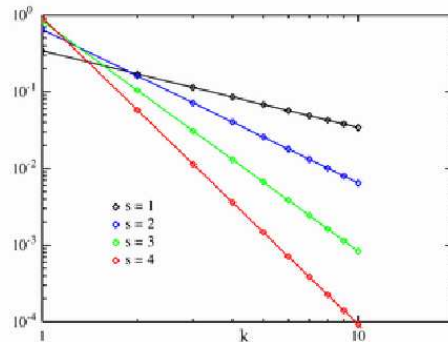
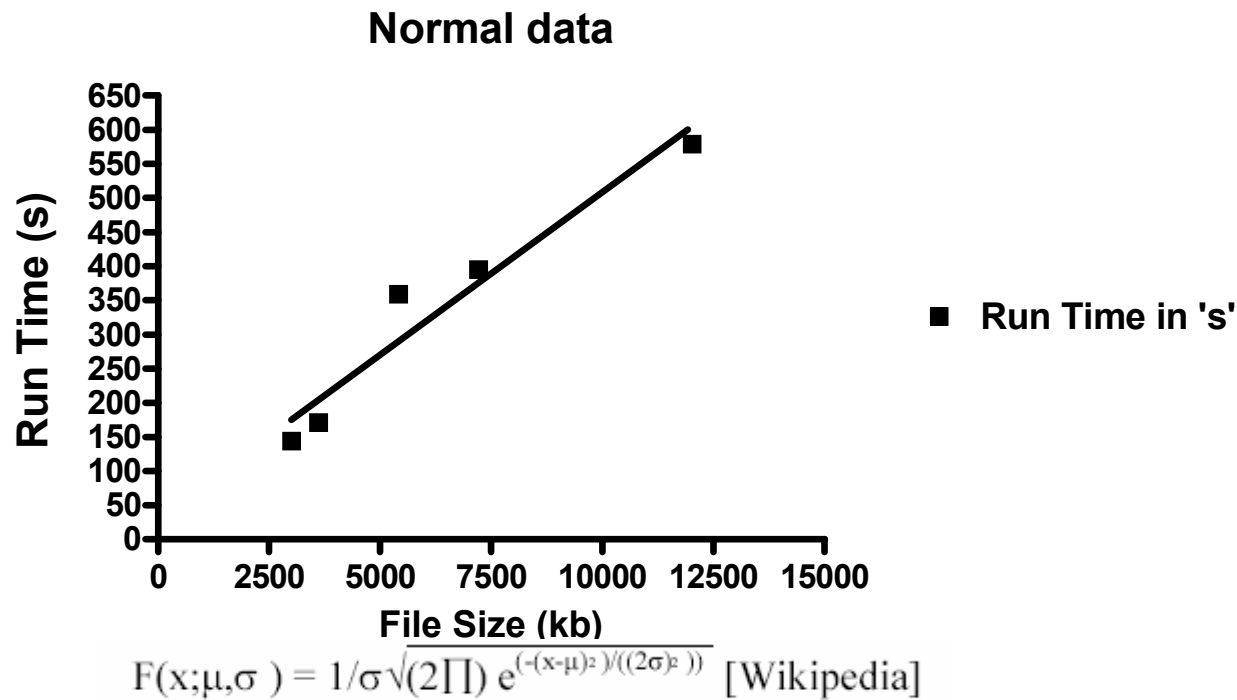


Figure 11 Zipf Distribution [Wikipedia]

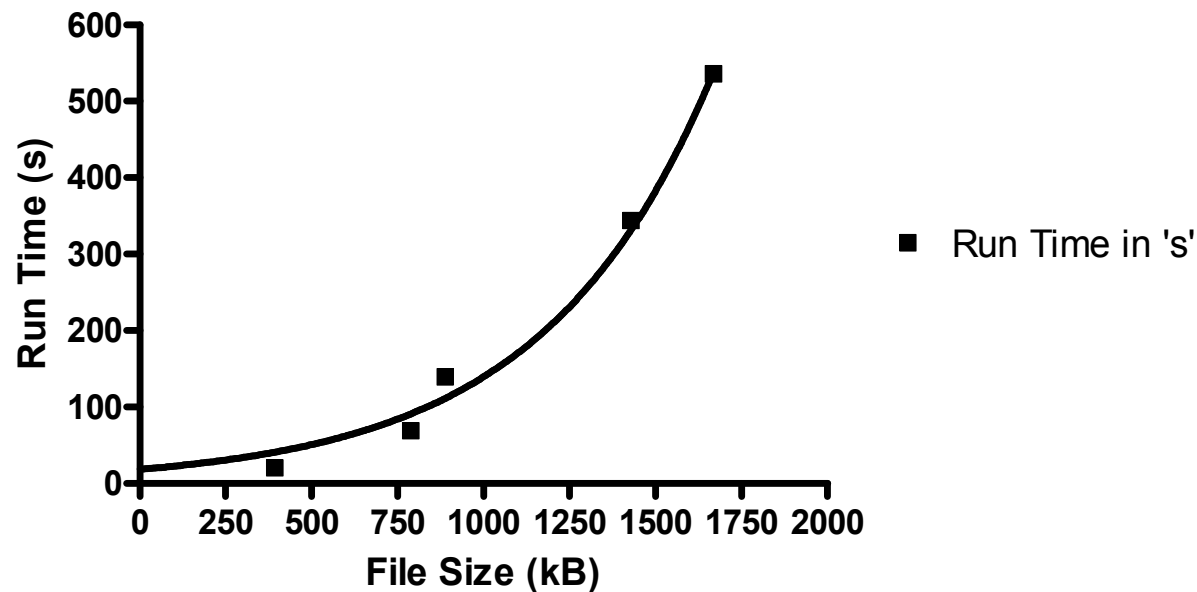
Running time of the Lattice algorithm on Normal Unbinned data



As we increase the file size, the running time increases linearly. This distribution is therefore chosen for binning.

Run time of Lattice algorithm with Exponential data and no binning

Exponential data

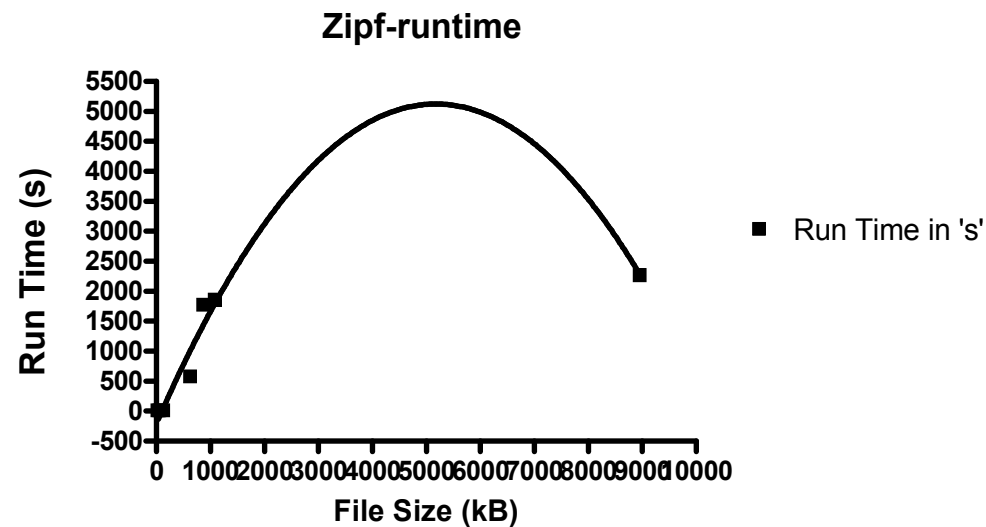


$$F(x; \lambda) = \lambda e^{-\lambda x} \text{ [Wikipedia]}$$

As we increase the file size, the running time increases exponentially.

Granules that fall under this distribution are chosen for binning experiments.

Running time of the Lattice algorithm on Zipf Unbinned data



$$F(k; s, N) = (1 / k^s) / (\sum_{n=1}^N 1 / n^s) \text{ [Wikipedia]}$$

Running time increases logarithmically.

Running time flattens out.



Results

- Zipf data performed better than normal and exponential
- With zipf-data, numbers are clustered around a few values (few items appear in most of the transactions).
- With remaining distributions, data is not clustered.
- Running the lattice algorithm with large Zipf datasets seems feasible.



Improving Running Time

- For non-Zipf data, running time of the algorithm is high
- For such data, it maybe desirable to derive trends about the data
- Key Idea: Use binning to transform non-Zipf data into a more tractable form



Binning Procedure

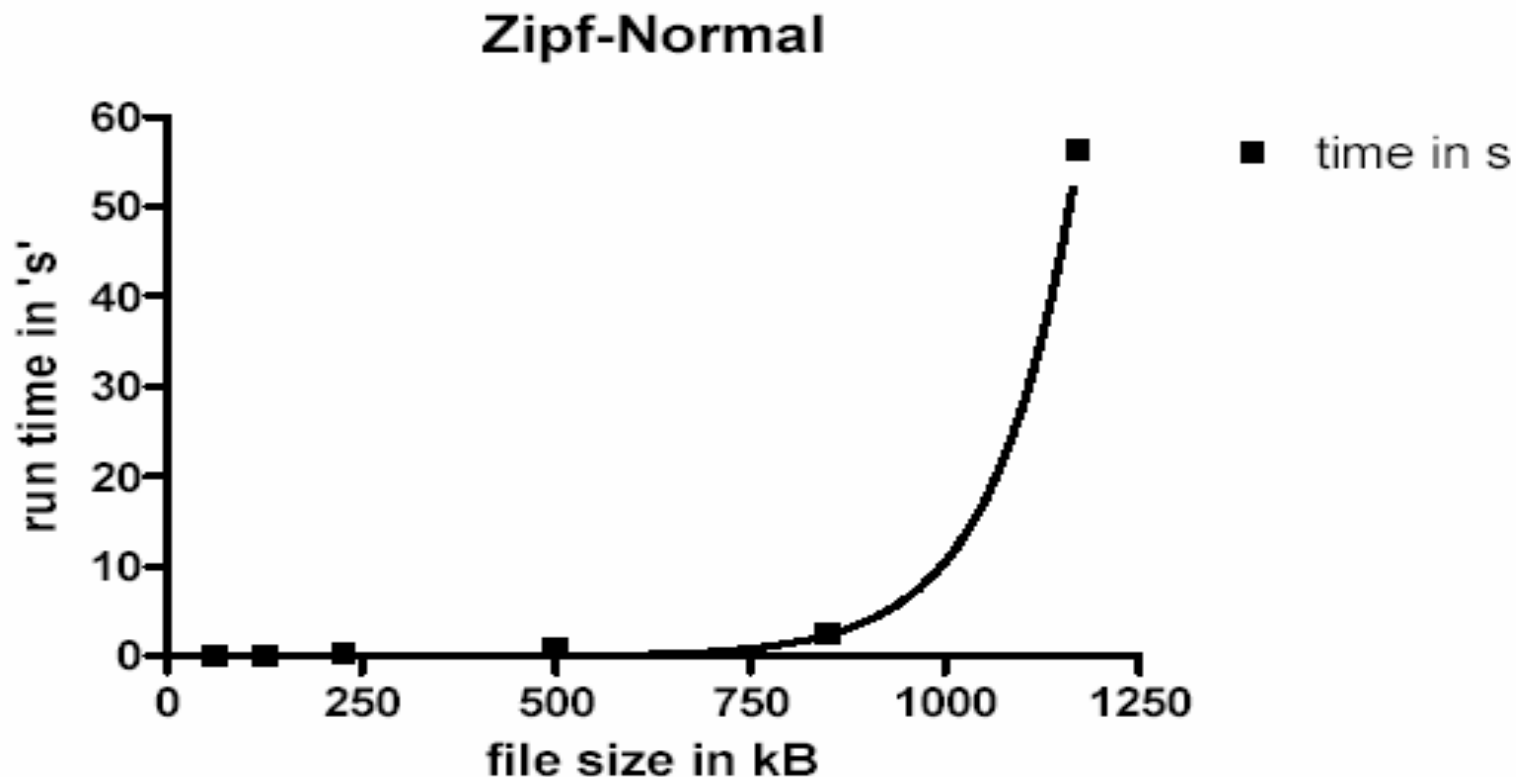
- For each granule, use the Chi-square test to see which distribution the granule matches closely
- For non-zipf granules:
 - Divide data into uniform sized bins and construct histogram
 - If frequency of a bin exceeds a threshold, then all data values for that bin are represented by the log of the bin's frequency
 - Otherwise, values are represented as is



Transforming Normal

- Draw a figure that shows how normal distribution is mapped to zipf distribution

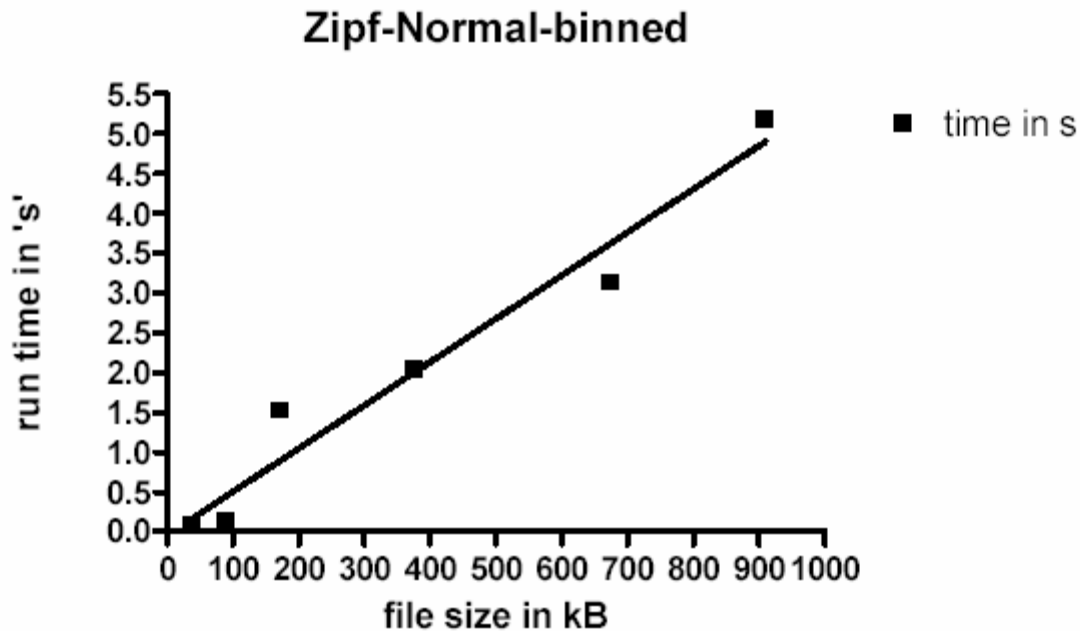
Running time of the Lattice algorithm on Zipf-Normal Unbinned data



Unbinned data runs in exponential time as file size increases

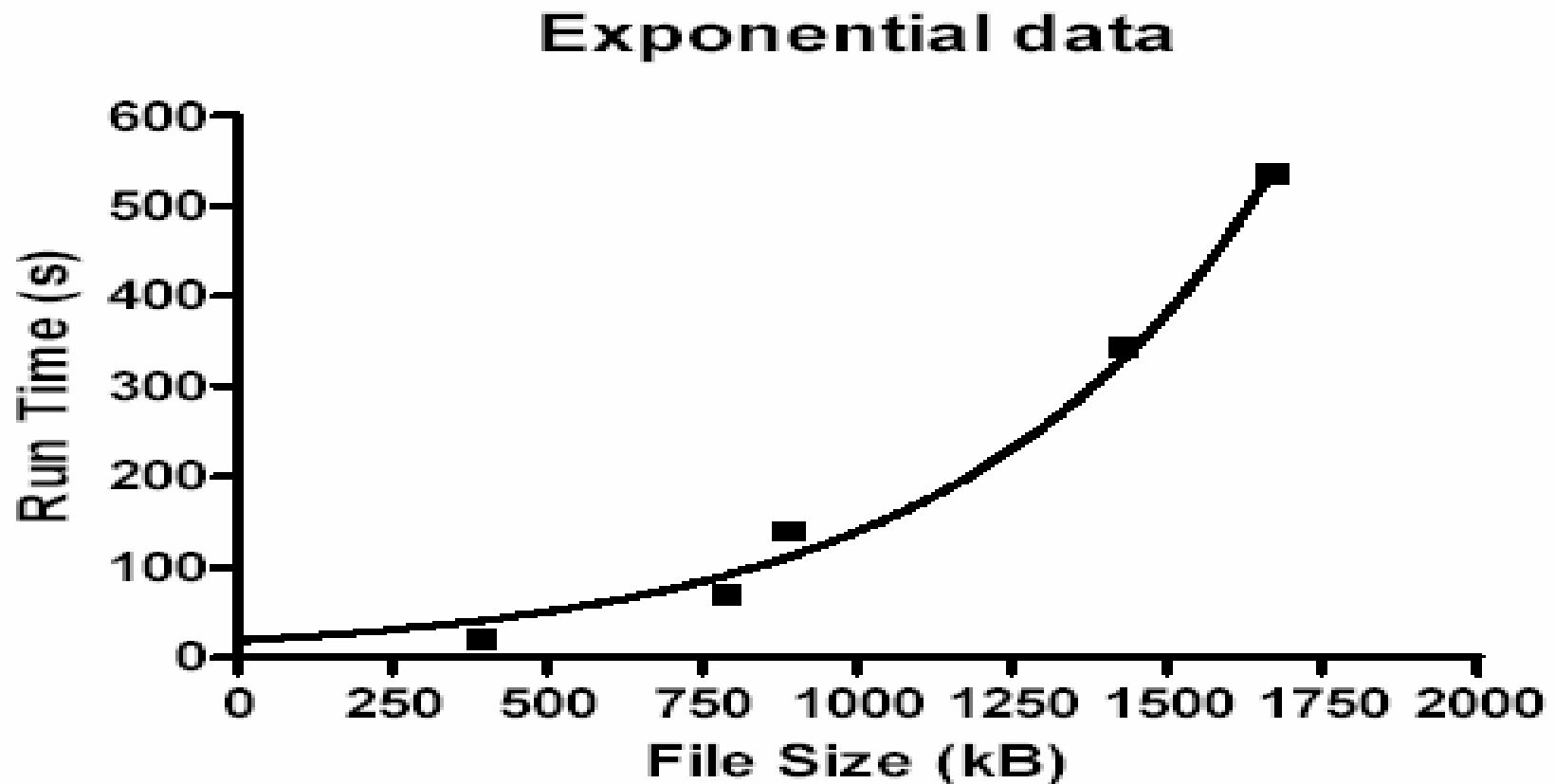
We will later bin the granules that are “normal”

Running time of the Lattice algorithm on Zipf Normal binned data



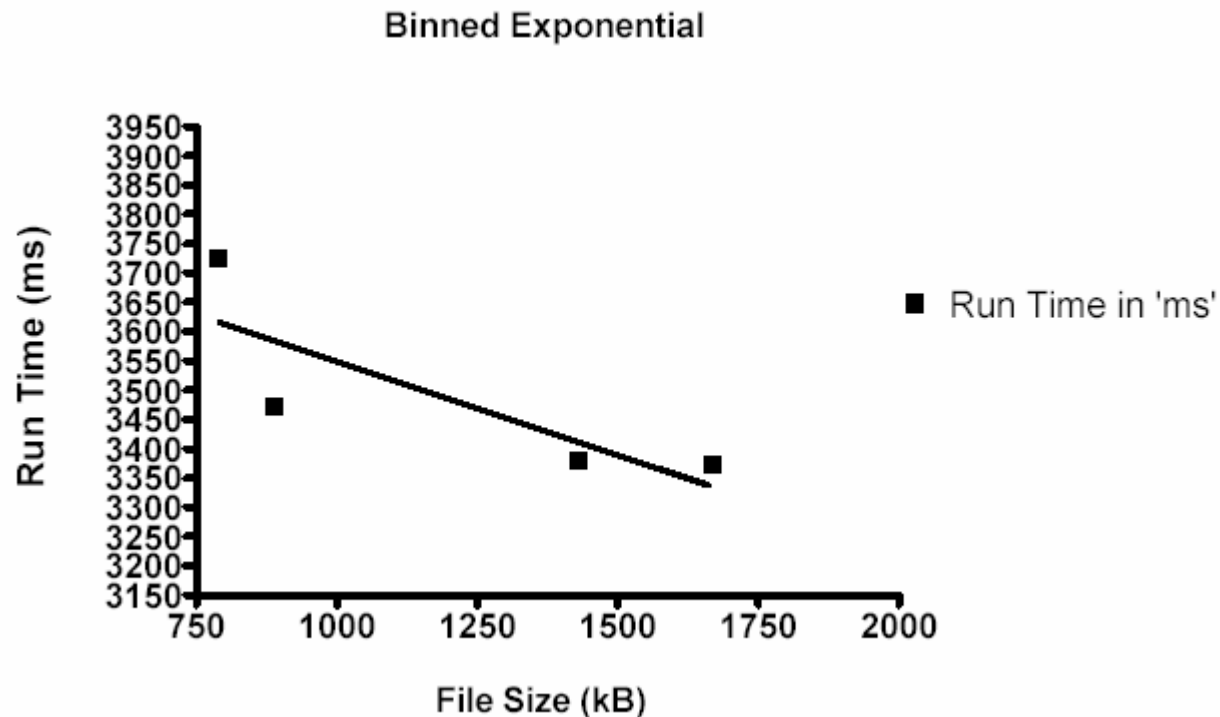
Run time increases linearly with increasing file size.

Running time of the Lattice algorithm on Unbinned Exponential data



Exponential growth is seen with exponential data as file size increases.

Running time of the Lattice algorithm on Exponential binned data



The running time slows down and flattens out as file size increases.



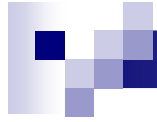
Conclusions

- Built upon the algorithm of Lin et. al. for mining closed frequent itemsets
- Identified issues with the algorithm and proposed solutions
- Developed an implementation and did a performance study
- Developed a novel binning mechanism to improve the running time for certain datasets



Future work

- Build upon the algorithm using it in an e-commerce application.
- Explore other binning techniques.
- Compare against other mining algorithms such as Charm, Closet, Pascal (that also mine for closed frequent itemsets).



Backup Slides



Result of Apriori

- Apriori provides good run-time performance when length is small.
- Performance is impacted by:
 - Pruning efficiency: Many frequently occurring patterns => pruning is less efficient.
If S consists of frequent itemset of length k , there could be upto $2^{|S|} - 2$ candidates of length $k+1$.
Computation is CPU bound.
 - Number of database scans: proportional to the length of the longest frequent itemset . In real world, itemsets of length 30 or higher is typical



Phase Two

- Procedure ExpandFreqItemSet(Nodes, minsup)
- For every node $li \in \text{Nodes}$
 - NewNodes = \emptyset , $I = li$
 - For each sibling lj after li in Nodes
- 1.2.1 $I = li \cup lj$ and $B_{\text{comb}} = B(li) \cap B(lj)$
- 1.2.2 If $B_{\text{comb}} > \text{minSup}$
- 1.2.2.1 Add $I \times B_{\text{comb}}$ to the NewNode
- 1.2.3 Else add li 's parents to the NewNode
- 1.3 $F = F \cup I$
- If NewNodes $\neq \emptyset$, then ExpandFreqItemSet(NewNodes, minsup)



Overall Procedure

- Main()
- $C = \{ \}$ // set of closed frequent itemsets
- $F = \{ \}$ // set of frequent itemsets
- Construct attribute value lattice (i.e., Phase one)
- Expand frequent itemsets (i.e., Phase 2)
- For every node $li \in F$, add the ancestor set of li to C




Binning Experiments

- Studied two types of data
 - Exponential distribution
 - Mixed data: Granules are a mix of Zipf and Exponential distributions



Pre-Apriori : Example (Generating candidates of length 2)

	$\langle m \rangle$	$\langle n \rangle$	$\langle o \rangle$
$\langle m \rangle$	$\langle mm \rangle$	$\langle mn \rangle$	$\langle mo \rangle$
$\langle n \rangle$	$\langle nm \rangle$	$\langle nn \rangle$	$\langle no \rangle$
$\langle o \rangle$	$\langle om \rangle$	$\langle on \rangle$	$\langle oo \rangle$



Apriori – Example (contd.) (Generating candidates of length 2)

	<m>	<n>	<o>
<m>		<mn>	<mo>
<n>			<no>
<o>			



Preliminaries: Poset

- A binary relation \leq that satisfies reflexive, symmetric, and transitive relationships on a set P is said to be a **partially ordered set** (poset)
 - **Reflexive:** $a \leq a$
 - **Symmetric:** $a \leq b \wedge b \leq a \Rightarrow a = b$
 - **Transitive:** $a \leq b \wedge b \leq c \Rightarrow a \leq c$
- Example: (\mathbb{N}, \leq) , where \mathbb{N} is the set of natural numbers



Preliminaries: Lattice

- A **poset** is a *lattice* if all non-empty finite subsets have a *greatest lower bound* and a *least upper bound*.
- Let $S \subseteq P$ and $u, l \in P$. Then:
 - u is the least upper bound if and only if, $\forall s \in S, s \leq u$
 - l is the greatest lower bound if and only if, $\forall s \in S, l \leq s$



Putting It All Together

- The set of granules from the database with the \subseteq relationship defined on the bitmaps is a poset
- Set of granules for the frequent itemsets under \subseteq relation is a lattice
 - Greatest lower bound: \emptyset is a subset of all frequent itemsets
 - Least upper bound: Set formed by taking the union of all the granules

Sample Attribute Value Lattice

