Advanced Frequent Pattern Mining
&
Locality Sensitive Hashing

Huan Sun, CSE@The Ohio State University
**Sequence Mining: Description**

**Input**
- A database $D$ of sequences called *data-sequences*, in which:
  - $I=\{i_1, i_2, \ldots, i_n\}$ is the set of items
  - each sequence is a list of transactions ordered by transaction-time
  - each transaction consists of fields: sequence-id, transaction-id, transaction-time and a set of items.

<table>
<thead>
<tr>
<th>Sequence-Id</th>
<th>Transaction Time</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>1</td>
<td>Ringworld</td>
</tr>
<tr>
<td>C1</td>
<td>2</td>
<td>Foundation</td>
</tr>
<tr>
<td>C1</td>
<td>15</td>
<td>Ringworld Engineers, Second Foundation</td>
</tr>
<tr>
<td>C2</td>
<td>1</td>
<td>Foundation, Ringworld</td>
</tr>
<tr>
<td>C2</td>
<td>20</td>
<td>Foundation and Empire</td>
</tr>
<tr>
<td>C2</td>
<td>50</td>
<td>Ringworld Engineers</td>
</tr>
</tbody>
</table>
## Sequential Pattern and Sequential Pattern Mining

- **Sequential pattern mining**: Given a set of sequences, find the complete set of frequent subsequences (i.e., satisfying the min_sup threshold)

### Sequence database

<table>
<thead>
<tr>
<th>SID</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>&lt;a(abc)(ac)d(cf)&gt;</td>
</tr>
<tr>
<td>20</td>
<td>&lt;(ad)c(bc)(ae)&gt;</td>
</tr>
<tr>
<td>30</td>
<td>&lt;(ef)(ab)(df)cb&gt;</td>
</tr>
<tr>
<td>40</td>
<td>&lt;eg(af)cbbc&gt;</td>
</tr>
</tbody>
</table>

- A **sequence** is a sequence of elements, where:
  - An **element** may contain a set of **items** (also called **events**)
  - Items within an element are unordered and we list them alphabetically

- `<a(bc)dc>` is a **subsequence** of `<a(abc)(ac)d(cf)>`

---

**Example**: Consider the sequence database:

- Sequence 10: `<a(abc)(ac)d(cf)>`
- Sequence 20: `<(ad)c(bc)(ae)>`
- Sequence 30: `<(ef)(ab)(df)cb>`
- Sequence 40: `<eg(af)cbbc>`

For these sequences, find all frequent subsequences using a min_sup threshold.
Sequential Pattern and Sequential Pattern Mining

- **Sequential pattern mining**: Given a set of sequences, find the complete set of frequent subsequences (i.e., satisfying the min_sup threshold)

A *sequence database*

<table>
<thead>
<tr>
<th>SID</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>&lt;a(abc)(ac)d(cf)&gt;</td>
</tr>
<tr>
<td>20</td>
<td>&lt;(ad)c(bc)(ae)&gt;</td>
</tr>
<tr>
<td>30</td>
<td>&lt;(ef)(ab)(df)c(b)&gt;</td>
</tr>
<tr>
<td>40</td>
<td>&lt;eg(af)cbc&gt;</td>
</tr>
</tbody>
</table>

Formal definition:

A sequence $\alpha = \langle a_1a_2\cdots a_n \rangle$ is called a subsequence of another sequence $\beta = \langle b_1b_2\cdots b_m \rangle$, and $\beta$ is a supersequence of $\alpha$, denoted as $\alpha \sqsubseteq \beta$, if there exist integers $1 \leq j_1 < j_2 < \cdots < j_n \leq m$ such that $a_1 \subseteq b_{j_1}$, $a_2 \subseteq b_{j_2}$, ..., $a_n \subseteq b_{j_n}$. For example, if $\alpha = \langle ab \rangle$ and $\beta = \langle (abc), (de) \rangle$, where $a$, $b$, $c$, $d$, and $e$ are items, then $\alpha$ is a subsequence of $\beta$ and $\beta$ is a supersequence of $\alpha$. 

*A(bc)dc* is a *subsequence* of *a(abc)(ac)d(cf)*
Sequential Pattern and Sequential Pattern Mining

- **Sequential pattern mining:** Given a set of sequences, find the complete set of frequent subsequences (i.e., satisfying the min_sup threshold)

A *sequence database*

<table>
<thead>
<tr>
<th>SID</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>&lt;a(abc)(ac)d(cf)&gt;</td>
</tr>
<tr>
<td>20</td>
<td>&lt;(ad)c(bc)(ae)&gt;</td>
</tr>
<tr>
<td>30</td>
<td>&lt;(ef)(ab)(df)cb&gt;</td>
</tr>
<tr>
<td>40</td>
<td>&lt;eg(af)cbc&gt;</td>
</tr>
</tbody>
</table>

A *sequence*: <(ef)(ab)(df)c|b>

- An **element** may contain a set of *items* (also called *events*)
- Items within an element are unordered and we list them alphabetically

&lt;a(bc)dc&gt; is a **subsequence** of &lt;a(abc)(ac)d(cf)&gt;

- Given *support threshold* min_sup = 2, &lt;(ab)c&gt; is a **sequential pattern**
A Basic Property of Sequential Patterns: Apriori

- A basic property: Apriori (Agrawal & Sirkant’94)
  - If a sequence $S$ is not frequent
  - Then none of the super-sequences of $S$ is frequent
  - E.g, $\langle hb \rangle$ is infrequent $\implies$ so do $\langle hab \rangle$ and $\langle (ah)b \rangle$
GSP (Generalized Sequential Patterns): Apriori-Based Sequential Pattern Mining

- Initial candidates: All 8-singleton sequences
  - \(<a>, <b>, <c>, <d>, <e>, <f>, <g>, <h>\)
- Scan DB once, count support for each candidate

`min_sup = 2`

<table>
<thead>
<tr>
<th>Cand.</th>
<th>sup</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;a&gt;)</td>
<td>3</td>
</tr>
<tr>
<td>(&lt;b&gt;)</td>
<td>5</td>
</tr>
<tr>
<td>(&lt;c&gt;)</td>
<td>4</td>
</tr>
<tr>
<td>(&lt;d&gt;)</td>
<td>3</td>
</tr>
<tr>
<td>(&lt;e&gt;)</td>
<td>3</td>
</tr>
<tr>
<td>(&lt;f&gt;)</td>
<td>2</td>
</tr>
<tr>
<td>(&lt;g&gt;)</td>
<td>1</td>
</tr>
<tr>
<td>(&lt;h&gt;)</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SID</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>(&lt;bd)cb(ac)&gt;</td>
</tr>
<tr>
<td>20</td>
<td>(&lt;bf)(ce)b(fg)&gt;</td>
</tr>
<tr>
<td>30</td>
<td>(&lt;ah)(bf)abf&gt;</td>
</tr>
<tr>
<td>40</td>
<td>(&lt;be)(ce)d&gt;</td>
</tr>
<tr>
<td>50</td>
<td>(&lt;a(bd)bcb(ade)&gt;</td>
</tr>
</tbody>
</table>
GSP (Generalized Sequential Patterns): Apriori-Based Sequential Pattern Mining

- Initial candidates: All 8-singleton sequences
  - \(<a>\), \(<b>\), \(<c>\), \(<d>\), \(<e>\), \(<f>\), \(<g>\), \(<h>\)
- Scan DB once, count support for each candidate
- Generate length-2 candidate sequences

<table>
<thead>
<tr>
<th>SID</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>(&lt;(bd)cb(ac)&gt;)</td>
</tr>
<tr>
<td>20</td>
<td>(&lt;(bf)(ce)b(fg)&gt;)</td>
</tr>
<tr>
<td>30</td>
<td>(&lt;(ah)(bf)abf&gt;)</td>
</tr>
<tr>
<td>40</td>
<td>(&lt;(be)(ce)d&gt;)</td>
</tr>
<tr>
<td>50</td>
<td>(&lt;a(bd)bcb(ade)&gt;)</td>
</tr>
</tbody>
</table>

GSP (Generalized Sequential Patterns): Srikant & Agrawal @ EDBT’96

How?
GSP (Generalized Sequential Patterns): Apriori-Based Sequential Pattern Mining

- Initial candidates: All 8-singleton sequences
  - <a>, <b>, <c>, <d>, <e>, <f>, <g>, <h>
- Scan DB once, count support for each candidate
- Generate length-2 candidate sequences

<table>
<thead>
<tr>
<th>SID</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>&lt;(bd)cb(ac)&gt;</td>
</tr>
<tr>
<td>20</td>
<td>&lt;(bf)(ce)b(fg)&gt;</td>
</tr>
<tr>
<td>30</td>
<td>&lt;(ah)(bf)abf&gt;</td>
</tr>
<tr>
<td>40</td>
<td>&lt;(be)(ce)d&gt;</td>
</tr>
<tr>
<td>50</td>
<td>&lt;a(bd)bcb(ade)&gt;</td>
</tr>
</tbody>
</table>

Why?
GSP (Generalized Sequential Patterns): Apriori-Based Sequential Pattern Mining

- Initial candidates: All 8-singleton sequences
  - \(<a>, <b>, <c>, <d>, <e>, <f>, <g>, <h>\)
- Scan DB once, count support for each candidate
- Generate length-2 candidate sequences

<table>
<thead>
<tr>
<th>SID</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>(&lt;(bd)cb(ac)&gt;)</td>
</tr>
<tr>
<td>20</td>
<td>(&lt;(bf)(ce)b(fg)&gt;)</td>
</tr>
<tr>
<td>30</td>
<td>(&lt;(ah)(bf)abf&gt;)</td>
</tr>
<tr>
<td>40</td>
<td>(&lt;(be)(ce)d&gt;)</td>
</tr>
<tr>
<td>50</td>
<td>(&lt;a(bd)bcb(ade)&gt;)</td>
</tr>
</tbody>
</table>

Why?
GSP (Generalized Sequential Patterns): Apriori-Based Sequential Pattern Mining

- Initial candidates: All 8-singleton sequences
  - \(<a>, <b>, <c>, <d>, <e>, <f>, <g>, <h>\)
- Scan DB once, count support for each candidate
- Generate length-2 candidate sequences

<table>
<thead>
<tr>
<th>SID</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>(&lt;(bd)cb(ac)&gt;)</td>
</tr>
<tr>
<td>20</td>
<td>(&lt;(bf)(ce)b(fg)&gt;)</td>
</tr>
<tr>
<td>30</td>
<td>(&lt;(ah)(bf)abf&gt;)</td>
</tr>
<tr>
<td>40</td>
<td>(&lt;(be)(ce)d&gt;)</td>
</tr>
<tr>
<td>50</td>
<td>(&lt;a(bd)bcb(ade)&gt;)</td>
</tr>
</tbody>
</table>

- Without Apriori pruning: (8 singletons) \(8 \times 8 + 8 \times 7/2 = 92\) length-2 candidates
- With pruning, length-2 candidates: \(36 + 15 = 51\)
GSP Mining and Pruning

5th scan: 1 cand. 1 length-5 seq. pat.

4th scan: 8 cand. 7 length-4 seq. pat.

3rd scan: 46 cand. 20 length-3 seq. pat. 20 cand. not in DB at all

2nd scan: 51 cand. 19 length-2 seq. pat. 10 cand. not in DB at all

1st scan: 8 cand. 6 length-1 seq. pat.

Candidates cannot pass min_sup threshold

Candidates not in DB

min_sup = 2

<table>
<thead>
<tr>
<th>SID</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>(bd)cb(ac)</td>
</tr>
<tr>
<td>20</td>
<td>(bf)(ce)b(fg)</td>
</tr>
<tr>
<td>30</td>
<td>(ah)(bf)abf</td>
</tr>
<tr>
<td>40</td>
<td>(be)(ce)d</td>
</tr>
<tr>
<td>50</td>
<td>a(bd)bcb(ade)</td>
</tr>
</tbody>
</table>
GSP Mining and Pruning

5th scan: 1 cand. 1 length-5 seq. pat.
4th scan: 8 cand. 7 length-4 seq. pat.
3rd scan: 46 cand. 20 length-3 seq. pat. 20 cand. not in DB at all
2nd scan: 51 cand. 19 length-2 seq. pat. 10 cand. not in DB at all
1st scan: 8 cand. 6 length-1 seq. pat.

- Repeat (for each level (i.e., length-k))
  - Scan DB to find length-k frequent sequences
  - Generate length-(k+1) candidate sequences from length-k frequent sequences using Apriori
  - set k = k+1
- Until no frequent sequence or no candidate can be found

**Candidates cannot pass min_sup threshold**

**Candidates not in DB**

**min_sup = 2**

<table>
<thead>
<tr>
<th>SID</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>(bd)cb(ac)</td>
</tr>
<tr>
<td>20</td>
<td>(bf)(ce)b(fg)</td>
</tr>
<tr>
<td>30</td>
<td>(ah)(bf)abf</td>
</tr>
<tr>
<td>40</td>
<td>(be)(ce)d</td>
</tr>
<tr>
<td>50</td>
<td>a(bd)bcb(ade)</td>
</tr>
</tbody>
</table>
GSP: Algorithm

- **Phase 1:**
  - Scan over the database to identify all the frequent items, i.e., 1-element sequences

- **Phase 2:**
  - Iteratively scan over the database to discover all frequent sequences. Each iteration discovers all the sequences with the same length.
  - In the iteration to generate all k-sequences
  - Generate the set of all candidate k-sequences, \( C_k \), by joining two \((k-1)\)-sequences
    - Prune the candidate sequence if any of its \( k-1 \) subsequences is not frequent
    - Scan over the database to determine the support of the remaining candidate sequences
  - Terminate when no more frequent sequences can be found

A detailed example illustration:

http://simpledatamining.blogspot.com/2015/03/generalized-sequential-pattern-gsp.html
Bottlenecks of GSP

- A huge set of candidates could be generated
  - 1,000 frequent length-1 sequences generate length-2 candidates!
    \[ 1000 \times 1000 + \frac{1000 \times 999}{2} = 1,499,500 \]
- Multiple scans of database in mining

- Real challenge: mining long sequential patterns
  - An exponential number of short candidates
  - A length-100 sequential pattern needs \(10^{30}\) candidate sequences!
    \[ \sum_{i=1}^{100} \binom{100}{i} = 2^{100} - 1 \approx 10^{30} \]
GSP: Optimization Techniques

- Applied to phase 2: computation-intensive

- Technique 1: the hash-tree data structure
  - Used for counting candidates to reduce the number of candidates that need to be checked
    - Leaf: a list of sequences
    - Interior node: a hash table

- Technique 2: data-representation transformation
  - From horizontal format to vertical format

<table>
<thead>
<tr>
<th>Transaction-Time</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1, 2</td>
</tr>
<tr>
<td>25</td>
<td>4, 6</td>
</tr>
<tr>
<td>45</td>
<td>3</td>
</tr>
<tr>
<td>50</td>
<td>1, 2</td>
</tr>
<tr>
<td>65</td>
<td>3</td>
</tr>
<tr>
<td>90</td>
<td>2, 4</td>
</tr>
<tr>
<td>95</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Item</th>
<th>Times</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>→ 10 → 50 → NULL</td>
</tr>
<tr>
<td>2</td>
<td>→ 10 → 50 → 90 → NULL</td>
</tr>
<tr>
<td>3</td>
<td>→ 45 → 65 → NULL</td>
</tr>
<tr>
<td>4</td>
<td>→ 25 → 90 → NULL</td>
</tr>
<tr>
<td>5</td>
<td>→ NULL</td>
</tr>
<tr>
<td>6</td>
<td>→ 25 → 95 → NULL</td>
</tr>
<tr>
<td>7</td>
<td>→ NULL</td>
</tr>
</tbody>
</table>
**Problems in the GSP Algorithm**
- Multiple database scans
- Complex hash structures with poor locality
- Scale up linearly as the size of dataset increases

**SPADE: Sequential PAttern Discovery using Equivalence classes**
- Use a vertical id-list database
- Prefix-based equivalence classes
- Frequent sequences enumerated through simple temporal joins
- Lattice-theoretic approach to decompose search space

**Advantages of SPADE**
- 3 scans over the database
- Potential for in-memory computation and parallelization

Paper Link:
MMDS Secs. 3.2-3.4.


FINDING SIMILAR ITEMS

Slides also adapted from Prof. Srinivasan Parthasarathy @OSU
Task: Finding Similar Documents

- **Goal:** Given a large number \((N \text{ in the millions or billions})\) of documents, find “near duplicate” pairs

- **Applications:**
  - Mirror websites, or approximate mirrors \(\rightarrow\) remove duplicates
  - Similar news articles at many news sites \(\rightarrow\) cluster
Task: Finding Similar Documents

- **Goal**: Given a large number \( N \) in the millions or billions) of documents, find “near duplicate” pairs

- **Applications:**
  - Mirror websites, or approximate mirrors → remove duplicates
  - Similar news articles at many news sites → cluster

What are the challenges?
Task: Finding Similar Documents

- **Goal:** Given a large number ($N$ in the millions or billions) of documents, find “near duplicate” pairs

- **Applications:**
  - Mirror websites, or approximate mirrors $\rightarrow$ remove duplicates
  - Similar news articles at many news sites $\rightarrow$ cluster

- **Problems:**
  - Many small pieces of one document can appear out of order in another
  - Too many documents to compare all pairs
  - Documents are so large or so many (scale issues)

Two Essential Steps for Similar Docs

1. **Shingling**: Convert documents to sets

2. **Min-Hashing**: Convert large sets to short signatures, while preserving similarity

   Host of follow up applications
   
   e.g. Similarity Search
   
   Data Placement
   
   Clustering etc.

The Big Picture

The set of strings of length $k$ that appear in the document

**Signatures:**
short integer vectors that represent the sets, and reflect their similarity

Similarity Search
Data Placement
Clustering etc.
SHINGLING

Step 1: **Shingling**: Convert documents to sets

The set of strings of length $k$ that appear in the document
Step 1: *Shingling*: Convert documents to sets

Simple approaches:
- Document = set of words appearing in document
- Document = set of “important” words
- Don’t work well for this application. Why?

Need to account for ordering of words!
A different way: *Shingles!*

Documents as High-Dim Data
Define: Shingles

- A $k$-shingle (or $k$-gram) for a document is a sequence of $k$ tokens that appears in the doc.
  - Tokens can be characters, words or something else, depending on the application.
  - Assume tokens = characters for examples.
Define: Shingles

- A \textit{k-shingle} (or \textit{k-gram}) for a document is a sequence of \textit{k} tokens that appears in the doc.
  - Tokens can be \textit{characters}, \textit{words} or something else, depending on the application.
  - Assume tokens = characters for examples.

- \textbf{Example:} \(k=2\); document \(D_1 = \text{abcab}\)
  - Set of 2-shingles: \(S(D_1) = \{ab, bc, ca\}\)
Define: Shingles

- A $k$-shingle (or $k$-gram) for a document is a sequence of $k$ tokens that appears in the doc
  - Tokens can be characters, words or something else, depending on the application
  - Assume tokens = characters for examples

- **Example:** $k=2$; document $D_1 = \text{abcab}$
  - Set of 2-shingles: $S(D_1) = \{ab, bc, ca\}$

- **Another option:** Shingles as a bag (multiset), count $ab$ twice: $S'(D_1) = \{ab, bc, ca, ab\}$

Shingles: How to treat white-space chars?

Example 3.4: If we use \( k = 9 \), but eliminate whitespace altogether, then we would see some lexical similarity in the sentences “The plane was ready for touch down”. and “The quarterback scored a touchdown”. However, if we retain the blanks, then the first has shingles `touch dow and ouch down`, while the second has `touchdown`. If we eliminated the blanks, then both would have `touchdown`. □

It makes sense to replace any sequence of one or more white-space characters (blank, tab, newline, etc.) by a single blank.

This way distinguishes shingles that cover two or more words from those that do not.
How to choose K?

- Documents that have lots of shingles in common have similar text, even if the text appears in different order.

- **Caveat:** You must pick $k$ large enough, or most documents will have most shingles.
  - $k = 5$ is OK for short documents.
  - $k = 10$ is better for long documents.
Compressing Shingles

- To **compress long shingles**, we can **hash** them to (say) 4 bytes
  - Like a Code Book
  - If #shingles manageable → Simple dictionary suffices

  e.g., 9-shingle => bucket number $[0, 2^{32} - 1]$
  (using 4 bytes instead of 9)
Compressing Shingles

- To **compress long shingles**, we can **hash** them to (say) 4 bytes
  - Like a Code Book
  - If #shingles manageable $\rightarrow$ Simple dictionary suffices

- **Doc represented by the set of hash/dict. values of its $k$-shingles**
  - **Idea:** Two documents could appear to have shingles in common, when the hash-values were shared
Compressing Shingles

- To compress long shingles, we can hash them to (say) 4 bytes
  - Like a Code Book
  - If #shingles manageable → Simple dictionary suffices

- Doc represented by the set of hash/dict. values of its $k$-shingles

- Example: $k=2$; document $D_1 = abcab$
  - Set of 2-shingles: $S(D_1) = \{ab, bc, ca\}$
  - Hash the singles: $h(D_1) = \{1, 5, 7\}$
Similarity Metric for Shingles

- **Document** $D_1$ is a set of its $k$-shingles $C_1 = S(D_1)$

- Equivalently, each document is a 0/1 vector in the space of $k$-shingles
  - Each unique shingle is a dimension
  - Vectors are very sparse

- **A natural similarity measure is the Jaccard similarity:**

$$sim(D_1, D_2) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|}$$
Motivation for Minhash/LSH

- Suppose we need to find similar documents among $N = 1$ million documents.

- Naïvely, we would have to compute pairwise Jaccard similarities for every pair of docs.

  - $N(N - 1)/2 \approx 5 \times 10^{11}$ comparisons
  - At $10^5$ secs/day and $10^6$ comparisons/sec, it would take 5 days.

- For $N = 10$ million, it takes more than a year...
Step 2: **Minhashing**: Convert large variable length sets to short fixed-length signatures, while preserving similarity.
Many similarity problems can be formalized as finding subsets that have significant intersection.
Many similarity problems can be formalized as finding subsets that have significant intersection.

Encode sets using 0/1 (bit, boolean) vectors
- One dimension per element in the universal set

Interpret set intersection as bitwise AND, and set union as bitwise OR.
Many similarity problems can be formalized as finding subsets that have significant intersection.

- **Encode sets using 0/1 (bit, boolean) vectors**
  - One dimension per element in the universal set
- **Interpret set intersection as bitwise AND, and set union as bitwise OR**

**Example:** \( C_1 = 10111; C_2 = 10011 \)
- Size of intersection = 3; size of union = 4,
- **Jaccard similarity** (not distance) = 3/4
- **Distance:** \( d(C_1, C_2) = 1 \) – (Jaccard similarity) = 1/4

From Sets to Boolean Matrices

- **Rows** = elements (shingles)
- **Columns** = sets (documents)
  - 1 in row $e$ and column $s$ if and only if $e$ is a valid shingle of document represented by $s$
  - Column similarity is the Jaccard similarity of the corresponding sets (rows with value 1)
  - Typical matrix is sparse!

Note: Transposed Document Matrix

<table>
<thead>
<tr>
<th>Documents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 0</td>
</tr>
<tr>
<td>1 1 0 1</td>
</tr>
<tr>
<td>0 1 0 1</td>
</tr>
<tr>
<td>0 0 0 1</td>
</tr>
<tr>
<td>1 0 0 1</td>
</tr>
<tr>
<td>1 1 1 0</td>
</tr>
<tr>
<td>1 0 1 0</td>
</tr>
</tbody>
</table>
Outline: Finding Similar Columns

- **So far:**
  - A document $\rightarrow$ a set of shingles
  - Represent a set as a boolean vector in a matrix

![Matrix representation of documents and shingles](http://www.mmds.org)
Outline: Finding Similar Columns

- **So far:**
  - A documents → a set of shingles
  - Represent a set as a boolean vector in a matrix

- **Next goal:** Find similar columns while computing small signatures
  - Similarity of columns == similarity of signatures
Outline: Finding Similar Columns

Next Goal: Find similar columns, Small signatures

Naïve approach:

1) Signatures of columns: small summaries of columns
Outline: Finding Similar Columns

- Next Goal: Find similar columns, Small signatures

- Naïve approach:
  1) Signatures of columns: small summaries of columns
  2) Examine pairs of signatures to find similar columns
     - Essential: Similarities of signatures and columns are related
  3) Optional: Check that columns with similar signatures are really similar
Next Goal: Find similar columns, Small signatures

Naïve approach:
1) Signatures of columns: small summaries of columns
2) Examine pairs of signatures to find similar columns
   Essential: Similarities of signatures and columns are related
3) Optional: Check that columns with similar signatures are really similar

Warnings:
Comparing all pairs may take too much time: Job for LSH
   These methods can produce false negatives, and even false positives (if the optional check is not made)
Key idea: “hash” each column $C$ to a small signature $h(C)$, such that:

1. $h(C)$ is small enough that the signature fits in RAM
2. $\text{sim}(C_1, C_2)$ is the same as the “similarity” of signatures $h(C_1)$ and $h(C_2)$
Hashing Columns (Signatures) : LSH principle

- **Key idea:** “hash” each column $C$ to a small *signature* $h(C)$, such that:
  - (1) $h(C)$ is small enough that the signature fits in RAM
  - (2) $\text{sim}(C_1, C_2)$ is the same as the “similarity” of signatures $h(C_1)$ and $h(C_2)$

- **Goal:** Find a hash function $h(\cdot)$ such that:
  - If $\text{sim}(C_1, C_2)$ is high, then with high prob. $h(C_1) = h(C_2)$
  - If $\text{sim}(C_1, C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$
Hashing Columns (Signatures) : LSH principle

- **Key idea:** “hash” each column $C$ to a small *signature* $h(C)$, such that:
  - (1) $h(C)$ is small enough that the signature fits in RAM
  - (2) $\text{sim}(C_1, C_2)$ is the same as the “similarity” of signatures $h(C_1)$ and $h(C_2)$

- **Goal:** Find a hash function $h(\cdot)$ such that:
  - If $\text{sim}(C_1, C_2)$ is high, then with high prob. $h(C_1) = h(C_2)$
  - If $\text{sim}(C_1, C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$

- Hash docs into buckets. Expect that “most” pairs of near duplicate docs hash into the same bucket!
Goal: Find a hash function $h(\cdot)$ such that:

- if $\text{sim}(C_1, C_2)$ is high, then with high prob. $h(C_1) = h(C_2)$
- if $\text{sim}(C_1, C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$

Clearly, the hash function depends on the similarity metric:

- Not all similarity metrics have a suitable hash function
Min-Hashing

- Goal: Find a hash function $h(\cdot)$ such that:
  - if $\text{sim}(C_1, C_2)$ is high, then with high prob. $h(C_1) = h(C_2)$
  - if $\text{sim}(C_1, C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$

- Clearly, the hash function depends on the similarity metric:
  - Not all similarity metrics have a suitable hash function

- There is a suitable hash function for the Jaccard similarity: It is called Min-Hashing
Min-Hashing

- Imagine the rows of the boolean matrix permuted under random permutation $\pi$
Min-Hashing

Imagine the rows of the boolean matrix permuted under random permutation $\pi$

Define a “hash” function $h_\pi(C)$ = the index of the first (in the permuted order $\pi$) row in which column $C$ has value 1:

$$h_\pi(C) = \min_\pi \pi(C)$$
Min-Hashing

- Imagine the rows of the boolean matrix permuted under random permutation $\pi$

- Define a “hash” function $h_\pi(C) =$ the index of the first (in the permuted order $\pi$) row in which column $C$ has value 1:
  \[
  h_\pi(C) = \min_\pi \pi(C)
  \]

- Use several (e.g., 100) independent hash functions (that is, permutations) to create a signature of a column
Min-Hashing

- Imagine the rows of the boolean matrix permuted under random permutation $\pi$

- Define a “hash” function $h_{\pi}(C) = \text{the index of the first (in the permuted order } \pi) \text{ row in which column C has value 1}$:
  $$h_{\pi}(C) = \min_{\pi} \pi(C)$$

- Use several (e.g., 100) independent hash functions (that is, permutations) to create a signature of a column
Zoo example (shingle size $k=1$)

Universe $\rightarrow$ \{ dog, cat, lion, tiger, mouse\}

$\pi_1 \rightarrow$ [ cat, mouse, lion, dog, tiger]

$\pi_2 \rightarrow$ [ lion, cat, mouse, dog, tiger]

$A = \{ \text{mouse, lion} \}$
Zoo example (shingle size k=1)

Universe $\rightarrow$ \{ dog, cat, lion, tiger, mouse\}

$\pi_1 \rightarrow$ [ cat, mouse, lion, dog, tiger]

$\pi_2 \rightarrow$ [ lion, cat, mouse, dog, tiger]

$A = \{ \text{mouse, lion} \}$

$mh_1(A) = \min ( \pi_1 \{ \text{mouse, lion} \} ) = \text{mouse}$

$mh_2(A) = \min ( \pi_2 \{ \text{mouse, lion} \} ) = \text{lion}$
Min-Hashing Example

Permutation $\pi$ | Input matrix (Shingles x Documents) | Signature matrix $M$
---|---|---
2 | 1 0 1 0 | 2 1 2 1
3 | 1 0 0 1 |
7 | 0 1 0 1 |
6 | 0 1 0 1 |
1 | 0 1 0 1 |
5 | 1 0 1 0 |
4 | 1 0 1 0 |
Min-Hashing Example

Permutation \( \pi \) of the input matrix (Shingles x Documents)

Signature matrix \( M \)

2\textsuperscript{nd} element of the permutation is the first to map to a 1
Min-Hashing Example

<table>
<thead>
<tr>
<th>Permutation $\pi$</th>
<th>Input matrix (Shingles x Documents)</th>
<th>Signature matrix $M$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 3 7 6 1 5 4</td>
<td>1 0 1 0 1 0 0 1 0 0 1 0 0 1 0 1</td>
<td>2 1 2 1 2 1 2 1 4 1 1</td>
</tr>
</tbody>
</table>

Min-Hashing Example

Permutation $\pi$  

Input matrix (Shingles x Documents)  

Signature matrix $M$  

2nd element of the permutation is the first to map to a 1  

4th element of the permutation is the first to map to a 1  

Permutation $\pi$:  

| 2 | 4 | 3 | 2 | 7 | 1 | 6 | 3 | 1 | 6 | 5 | 7 | 4 | 5 |

Input matrix (Shingles x Documents):  

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th></th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th></th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Min-Hashing Example

#### Input matrix (Shingles x Documents)

<table>
<thead>
<tr>
<th>2</th>
<th>4</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

#### Signature matrix $M$

<table>
<thead>
<tr>
<th>2</th>
<th>1</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

---

Min-Hashing Example

Permutation \( \pi \)  

Input matrix (Shingles x Documents)

Signature matrix \( M \)

Note: Another (equivalent) way is to store row indexes or raw shingles (e.g. mouse, lion):

\[
\begin{array}{cccc}
1 & 5 & 1 & 5 \\
2 & 3 & 1 & 3 \\
6 & 4 & 6 & 4 \\
\end{array}
\]

2\textsuperscript{nd} element of the permutation is the first to map to a 1

4\textsuperscript{th} element of the permutation is the first to map to a 1

Min-Hash Signatures

- Pick $K=100$ random permutations of the rows
- Think of $\text{sig}(C)$ as a column vector
- $\text{sig}(C)[i] = \text{ according to the } i\text{-th permutation, the index of the first row that has a 1 in column } C$
  
  $$\text{sig}(C)[i] = \min (\pi_i(C))$$

- **Note:** The sketch (signature) of document $C$ is small ~$100$ bytes!

- We achieved our goal! We “compressed” long bit vectors into short signatures
Key Fact

For two sets $A$, $B$, and a min-hash function $mh_i()$:

$$Pr[mh_i(A) = mh_i(B)] = Sim(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Unbiased estimator for $Sim$ using $K$ hashes (notation policy – this is a different $K$ from size of shingle)

$$Sim(A, B) = \frac{1}{k} \sum_{i=1}^{k} I[mh_i(A) = mh_i(B)]$$
## Min-Hashing Example

<table>
<thead>
<tr>
<th>Permutation $\pi$</th>
<th>Input matrix (Shingles x Documents)</th>
<th>Signature matrix $M$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 4 3</td>
<td>1 0 1 0 1 0 1 0 1 0</td>
<td>2 1 2 1</td>
</tr>
<tr>
<td>3 2 4</td>
<td>1 0 0 0 1</td>
<td>2 1 4 1</td>
</tr>
<tr>
<td>7 1 7</td>
<td>0 1 0 1</td>
<td>1 2 1 2</td>
</tr>
<tr>
<td>6 3 2</td>
<td>0 1 0 1</td>
<td></td>
</tr>
<tr>
<td>1 6 6</td>
<td>0 1 0 1</td>
<td></td>
</tr>
<tr>
<td>5 7 1</td>
<td>1 0 1 0</td>
<td></td>
</tr>
<tr>
<td>4 5 5</td>
<td>1 0 1 0</td>
<td></td>
</tr>
</tbody>
</table>

**Similarities:**

<table>
<thead>
<tr>
<th>Col/Col Sig/Sig</th>
<th>1-3 2-4 1-2 3-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-3</td>
<td>0.75 0.75 0 0</td>
</tr>
<tr>
<td>2-4</td>
<td>0.67 1.00 0 0</td>
</tr>
</tbody>
</table>

The Min-Hash Property

- Choose a random permutation $\pi$
- Claim: $\Pr[h_\pi(C_1) = h_\pi(C_2)] = \text{sim}(C_1, C_2)$
- Why?

One of the two cols had to have 1 at position $y$. 

The Min-Hash Property

- Choose a random permutation $\pi$
- **Claim:** $\Pr[h_\pi(C_1) = h_\pi(C_2)] = \text{sim}(C_1, C_2)$
- **Why?**
  - Let $X$ be a doc (set of shingles), $y \in X$ is a shingle

One of the two cols had to have 1 at position $y$
The Min-Hash Property

- **Choose a random permutation** $\pi$
- **Claim:** $\Pr[h_{\pi}(C_1) = h_{\pi}(C_2)] = sim(C_1, C_2)$
- **Why?**
  - Let $X$ be a doc (set of shingles), $y \in X$ is a shingle
  - **Then:** $\Pr[\pi(y) = \min(\pi(X))] = 1/|X|$
    - It is equally likely that any $y \in X$ is mapped to the $\min$ element

One of the two cols had to have 1 at position $y$
The Min-Hash Property

- Choose a random permutation $\pi$
- **Claim:** $\Pr[h_{\pi}(C_1) = h_{\pi}(C_2)] = \text{sim}(C_1, C_2)$
- **Why?**
  - Let $X$ be a doc (set of shingles), $y \in X$ is a shingle
  - **Then:** $\Pr[\pi(y) = \min(\pi(X))] = 1/|X|$
    - It is equally likely that any $y \in X$ is mapped to the min element
  - Let $y$ be s.t. $\pi(y) = \min(\pi(C_1 \cup C_2))$
  - **Then either:**
    - $\pi(y) = \min(\pi(C_1))$ if $y \in C_1$, or
    - $\pi(y) = \min(\pi(C_2))$ if $y \in C_2$

One of the two cols had to have 1 at position $y$
The Min-Hash Property

- Choose a random permutation $\pi$
- Claim: $\Pr[h_\pi(C_1) = h_\pi(C_2)] = \text{sim}(C_1, C_2)$

Why?
- Let $X$ be a doc (set of shingles), $y \in X$ is a shingle
- Then: $\Pr[\pi(y) = \min(\pi(X))] = 1/|X|$  
  - It is equally likely that any $y \in X$ is mapped to the min element

- Let $y$ be s.t. $\pi(y) = \min(\pi(C_1 \cup C_2))$
- Then either: $\pi(y) = \min(\pi(C_1))$ if $y \in C_1$, or $\pi(y) = \min(\pi(C_2))$ if $y \in C_2$

- So the prob. that both are true is the prob. $y \in C_1 \cap C_2$
- $\Pr[\min(\pi(C_1))=\min(\pi(C_2))] = |C_1 \cap C_2|/|C_1 \cup C_2| = \text{sim}(C_1, C_2)$

One of the two cols had to have 1 at position $y$

The Min-Hash Property (Take 2: simpler proof)

- **Choose a random permutation** \( \pi \)
- **Claim:** \( \Pr[h_\pi(C_1) = h_\pi(C_2)] = \text{sim}(C_1, C_2) \)
- **Why?**
  - Given a set \( X \), the probability that any one element is the min-hash under \( \pi \) is \( 1/|X| \) \( \leftarrow (0) \)
    - It is equally likely that any \( y \in X \) is mapped to the min element
  - Given a set \( X \), the probability that one of any \( k \) elements is the min-hash under \( \pi \) is \( k/|X| \) \( \leftarrow (1) \)
  - For \( C_1 \cup C_2 \), the probability that any element is the min-hash under \( \pi \) is \( 1/|C_1 \cup C_2| \) \( \text{from (0)} \) \( \leftarrow (2) \)
  - For any \( C_1 \) and \( C_2 \), the probability of choosing the same min-hash under \( \pi \) is \( |C_1 \cap C_2|/|C_1 \cup C_2| \) \( \leftarrow \text{from (1) and (2)} \)
Similarity for Signatures

- We know: $\Pr[h_\pi(C_1) = h_\pi(C_2)] = \text{sim}(C_1, C_2)$
- Now generalize to multiple hash functions

- The similarity of two signatures is the fraction of the hash functions in which they agree

- Note: Because of the Min-Hash property, the similarity of columns is the same as the expected similarity of their signatures
Min-Hashing Example

Permutation $\pi$

<table>
<thead>
<tr>
<th>2</th>
<th>4</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

Input matrix (Shingles x Documents)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Signature matrix $M$

<table>
<thead>
<tr>
<th>2</th>
<th>1</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Similarities:

<table>
<thead>
<tr>
<th>Col/Col</th>
<th>Sig/Sig</th>
<th>1-3</th>
<th>2-4</th>
<th>1-2</th>
<th>3-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.75</td>
<td>0.75</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.67</td>
<td>1.00</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Min-Hash Signatures

- Pick $K=100$ random permutations of the rows
- Think of $\text{sig}(C)$ as a column vector
- $\text{sig}(C)[i] =$ according to the $i$-th permutation, the index of the first row that has a 1 in column $C$
  $$\text{sig}(C)[i] = \min (\pi_i(C))$$
- **Note:** The sketch (signature) of document $C$ is small $\sim 100$ bytes!

- We achieved our goal! We “compressed” long bit vectors into short signatures
Implementation Trick

- Permuting rows even once is prohibitive
- Approximate Linear Permutation Hashing
- Pick K independent hash functions (use a, b below)
  - Apply the idea on each column (document) for each hash function and get minhash signature

How to pick a random hash function h(x)?

Universal hashing:

\[ h_{a,b}(x)=((a \cdot x + b) \mod p) \mod N \]

where:
- a, b \ldots \text{random integers}
- p \ldots \text{prime number (p > N)}
Summary: 3 Steps

- **Shingling**: Convert documents to sets
  - We used hashing to assign each shingle an ID

- **Min-Hashing**: Convert large sets to short signatures, while preserving similarity
  - We used *similarity preserving hashing* to generate signatures with property $\Pr[h_\pi(C_1) = h_\pi(C_2)] = \text{sim}(C_1, C_2)$
  - We used hashing to get around generating random permutations

Outline: Finding Similar Columns

- **So far:**
  - Documents $\rightarrow$ Sets of shingles
  - Represent sets as boolean vectors in a matrix

- **Next goal:** Find similar columns while computing small signatures
  - Similarity of columns $==$ similarity of signatures

Next Goal: Find similar columns, Small signatures

Naïve approach:

1) Signatures of columns: small summaries of columns
2) Examine pairs of signatures to find similar columns
   Essential: Similarities of signatures and columns are related
3) Optional: Check that columns with similar signatures are really similar

Warnings:

Comparing all pairs may take too much time: Job for LSH
   These methods can produce false negatives, and even false positives (if the optional check is not made)
Hashing Columns (Signatures) : LSH principle

- **Key idea:** “hash” each column $C$ to a small *signature* $h(C)$, such that:
  - (1) $h(C)$ is small enough that the signature fits in RAM
  - (2) $\text{sim}(C_1, C_2)$ is the same as the “similarity” of signatures $h(C_1)$ and $h(C_2)$
**Hashing Columns (Signatures) : LSH principle**

- **Key idea:** “hash” each column \( C \) to a small *signature* \( h(C) \), such that:
  1. \( h(C) \) is small enough that the signature fits in RAM
  2. \( sim(C_1, C_2) \) is the same as the “similarity” of signatures \( h(C_1) \) and \( h(C_2) \)

- **Goal:** Find a hash function \( h(\cdot) \) such that:
  1. If \( sim(C_1, C_2) \) is high, then with high prob. \( h(C_1) = h(C_2) \)
  2. If \( sim(C_1, C_2) \) is low, then with high prob. \( h(C_1) \neq h(C_2) \)

- Hash docs into buckets. Expect that “most” pairs of near duplicate docs hash into the same bucket!

---

Min-Hashing

- **Goal:** Find a hash function \( h(\cdot) \) such that:
  - if \( \text{sim}(C_1, C_2) \) is high, then with high prob. \( h(C_1) = h(C_2) \)
  - if \( \text{sim}(C_1, C_2) \) is low, then with high prob. \( h(C_1) \neq h(C_2) \)

- **Clearly, the hash function depends on the similarity metric:**
  - Not all similarity metrics have a suitable hash function

- **There is a suitable hash function for the Jaccard similarity:** It is called **Min-Hashing**
Min-Hashing

- Imagine the rows of the boolean matrix permuted under random permutation $\pi$

- Define a “hash” function $h_\pi(C)$ = the index of the first (in the permuted order $\pi$) row in which column $C$ has value 1:
  \[ h_\pi(C) = \min_\pi \pi(C) \]

- Use several (e.g., 100) independent hash functions (that is, permutations) to create a signature of a column
Zoo example (shingle size k=1)

Universe  $\rightarrow$ { dog, cat, lion, tiger, mouse}
$\pi_1$  $\rightarrow$ [ cat, mouse, lion, dog, tiger]
$\pi_2$  $\rightarrow$ [ lion, cat, mouse, dog, tiger]

$A = \{ \text{mouse, lion} \}$

$mh_1(A) = \min ( \pi_1 \{ \text{mouse, lion} \}) = \text{mouse}$

$mh_2(A) = \min ( \pi_2 \{ \text{mouse, lion} \}) = \text{lion}$
Key Fact

For two sets $A$, $B$, and a min-hash function $mh_i()$:

$$Pr[mh_i(A) = mh_i(B)] = Sim(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Unbiased estimator for $Sim$ using $K$ hashes (notation policy – this is a different $K$ from size of shingle)

$$\hat{Sim}(A, B) = \frac{1}{k} \sum_{i=1:k} I[mh_i(A) = mh_i(B)]$$
Min-Hashing Example

Note: Another (equivalent) way is to store row indexes or raw shingles (e.g. mouse, lion):

\[ M = \begin{bmatrix}
2 & 1 & 2 & 1 \\
2 & 1 & 4 & 1 \\
1 & 2 & 1 & 2 \\
\end{bmatrix} \]

Input matrix (Shingles x Documents):

Permutation \( \pi \):

2\textsuperscript{nd} element of the permutation is the first to map to a 1

4\textsuperscript{th} element of the permutation is the first to map to a 1

The Min-Hash Property

- **Choose a random permutation** \(\pi\)

- **Claim:** \(\Pr[h_\pi(C_1) = h_\pi(C_2)] = \text{sim}(C_1, C_2)\)

- **Why?**
  - Let \(X\) be a doc (set of shingles), \(y \in X\) is a shingle
  - Then: \(\Pr[\pi(y) = \min(\pi(X))] = 1/|X|\)
    - It is equally likely that any \(y \in X\) is mapped to the min element
  - Let \(y\) be s.t. \(\pi(y) = \min(\pi(C_1 \cup C_2))\)
  - Then either: \(\pi(y) = \min(\pi(C_1))\) if \(y \in C_1\), or \(\pi(y) = \min(\pi(C_2))\) if \(y \in C_2\)
  - So the prob. that both are true is the prob. \(y \in C_1 \cap C_2\)
  - \(\Pr[\min(\pi(C_1))=\min(\pi(C_2))] = |C_1 \cap C_2| / |C_1 \cup C_2| = \text{sim}(C_1, C_2)\)

One of the two cols had to have 1 at position \(y\)
The Min-Hash Property (Take 2: simpler proof)

- **Choose a random permutation** $\pi$
- **Claim:** $\Pr[h_\pi(C_1) = h_\pi(C_2)] = \text{sim}(C_1, C_2)$
- **Why?**
  - Given a set $X$, the probability that any one element is the min-hash under $\pi$ is $1/|X|$ \(\leftarrow (0)\)
    - It is equally likely that any $y \in X$ is mapped to the $\text{min}$ element
  - Given a set $X$, the probability that one of any $k$ elements is the min-hash under $\pi$ is $k/|X|$ \(\leftarrow (1)\)
  - For $C_1 \cup C_2$, the probability that any element is the min-hash under $\pi$ is $1/|C_1 \cup C_2|$ (from 0) \(\leftarrow (2)\)
  - For any $C_1$ and $C_2$, the probability of choosing the same min-hash under $\pi$ is $|C_1 \cap C_2|/|C_1 \cup C_2|$ \(\leftarrow \) from (1) and (2)
We know: $\Pr[h_\pi(C_1) = h_\pi(C_2)] = \text{sim}(C_1, C_2)$

Now generalize to multiple hash functions

The *similarity of two signatures* is the fraction of the hash functions in which they agree

**Note:** Because of the Min-Hash property, the similarity of columns is the same as the expected similarity of their signatures

Min-Hashing Example

**Permutation**: $\pi$

<table>
<thead>
<tr>
<th>2</th>
<th>4</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

**Input matrix (Shingles x Documents)**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**Signature matrix $M$**

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>1</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

**Similarities**

<table>
<thead>
<tr>
<th>Col/Col</th>
<th>1-3</th>
<th>2-4</th>
<th>1-2</th>
<th>3-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sig/Sig</td>
<td>0.75</td>
<td>0.75</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0.67</td>
<td>1.00</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Min-Hash Signatures

- Pick $K=100$ random permutations of the rows
- Think of $\text{sig}(C)$ as a column vector
- $\text{sig}(C)[i] =$ according to the $i$-th permutation, the index of the first row that has a 1 in column $C$
  
  $$\text{sig}(C)[i] = \min (\pi_i(C))$$

- **Note:** The sketch (signature) of document $C$ is small $\sim 100$ bytes!

- **We achieved our goal!** We “compressed” long bit vectors into short signatures
Implementation Trick

- Permuting rows even once is prohibitive
- Approximate Linear Permutation Hashing
- Pick K independent hash functions (use a, b below)
  - Apply the idea on each column (document) for each hash function and get minhash signature

How to pick a random hash function $h(x)$?

Universal hashing:

$$h_{a,b}(x)=((a \cdot x + b) \mod p) \mod N$$

where:
- $a,b$ ... random integers
- $p$ ... prime number ($p > N$)
Summary: 3 Steps

- **Shingling**: Convert documents to sets
  - We used hashing to assign each shingle an ID

- **Min-Hashing**: Convert large sets to short signatures, while preserving similarity
  - We used similarity preserving hashing to generate signatures with property $\Pr[h_\pi(C_1) = h_\pi(C_2)] = \text{sim}(C_1, C_2)$
  - We used hashing to get around generating random permutations

Sequential Pattern Mining in Vertical Data Format: The SPADE Algorithm

- A sequence database is mapped to: <SID, EID>
- Grow the subsequences (patterns) one item at a time by Apriori candidate generation

Ref: SPADE (Sequential Pattern Discovery using Equivalent Class) [M. Zaki 2001]
PrefixSpan: A Pattern-Growth Approach

PrefixSpan Mining: Prefix Projections

- Step 1: Find length-1 sequential patterns
  - <a>, <b>, <c>, <d>, <e>, <f>

- Step 2: Divide search space and mine each projected DB
  - <a>-projected DB,
  - <b>-projected DB,
  - ...
  - <f>-projected DB, ...

Prefix and suffix

- Prefixes: <a>, <aa>, <a(ab)>, <a(abc)>, ...

- Suffix: Prefixes-based projection

PrefixSpan (Prefix-projected Sequential pattern mining)
Pei, et al. @TKDE’04

PrefixSpan
Mining: Prefix Projections

<table>
<thead>
<tr>
<th>SID</th>
<th>Sequence</th>
<th>Prefix</th>
<th>Suffix (Projection)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>&lt;a(abc)(ac)d(cf)&gt;</td>
<td>&lt;a&gt;</td>
<td>&lt;a(abc)(ac)d(cf)&gt;</td>
</tr>
<tr>
<td>20</td>
<td>&lt;ad)c(bc)(ae)&gt;</td>
<td>&lt;aa&gt;</td>
<td>&lt;_bc)(ac)d(cf)&gt;</td>
</tr>
<tr>
<td>30</td>
<td>&lt;ef)(ab)(df)cb&gt;</td>
<td>&lt;ab&gt;</td>
<td>&lt;_c)(ac)d(cf)&gt;</td>
</tr>
<tr>
<td>40</td>
<td>&lt;eg(af)cbbc&gt;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
PrefixSpan: Mining Prefix-Projected DBs

<table>
<thead>
<tr>
<th>SID</th>
<th>Sequence</th>
<th>min_sup = 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>&lt;a(abc)(ac)d(cf)&gt;</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>&lt;(ad)c(bc)(ae)&gt;</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>&lt;(ef)(ab)(df)cb&gt;</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>&lt;eg(af)cbc&gt;</td>
<td></td>
</tr>
</tbody>
</table>

Length-1 sequential patterns: <a>, <b>, <c>, <d>, <e>, <f>

Length-2 sequential patterns:
- <aa>, <ab>, <(ab)>, <ac>, <ad>, <af>
- <aa>, <ab>, <(ab)>, <ac>, <ad>, <af>

Major strength of PrefixSpan:
- No candidate subsequences to be generated
- Projected DBs keep shrinking
Consideration:
Pseudo-Projection vs. Physical Projection

- Major cost of PrefixSpan: Constructing projected DBs
  - Suffixes largely repeating in recursive projected DBs

- When DB can be held in main memory, use pseudo projection
  - No physically copying suffixes
  - Pointer to the sequence
  - Offset of the suffix
  - But if it does not fit in memory
    - Physical projection
  - Suggested approach:
    - Integration of physical and pseudo-projection
    - Swapping to pseudo-projection when the data fits in memory
CloSpan: Mining Closed Sequential Patterns

- A closed sequential pattern $s$: There exists no superpattern $s'$ such that $s' \supset s$, and $s'$ and $s$ have the same support.

- Which ones are closed? $\langle abc \rangle$: 20, $\langle abcd \rangle$: 20, $\langle abcde \rangle$: 15

- Why directly mine closed sequential patterns?
  - Reduce # of (redundant) patterns
  - Attain the same expressive power

- Property $P_1$: If $s \supset s'$, $s$ is closed iff two project DBs have the same size

- Explore Backward Subpattern and Backward Superpattern pruning to prune redundant search space

- Greatly enhances efficiency (Yan, et al., SDM’03)
CloSpan: When Two Projected DBs Have the Same Size

- If $s \subseteq s_1$, $s$ is closed iff two project DBs have the same size
- When two projected sequence DBs have the same size?
- Here is one example:

<table>
<thead>
<tr>
<th>ID</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&lt;afefbcg&gt;</td>
</tr>
<tr>
<td>2</td>
<td>&lt;afegb(ac)&gt;</td>
</tr>
<tr>
<td>3</td>
<td>&lt;(af)ea&gt;</td>
</tr>
</tbody>
</table>

$\text{min}_{\text{sup}} = 2$

- **Backward subpattern pruning**
- **Backward superpattern pruning**

Only need to keep size = 12 (including parentheses)
Chapter 7: Advanced Frequent Pattern Mining

- Mining Diverse Patterns
- Sequential Pattern Mining
- Constraint-Based Frequent Pattern Mining
- Graph Pattern Mining
- Pattern Mining Application: Mining Software Copy-and-Paste Bugs
- Summary
Constraint-Based Pattern Mining

- Why Constraint-Based Mining?
- Different Kinds of Constraints: Different Pruning Strategies
  - Constrained Mining with Pattern Anti-Monotonicity
  - Constrained Mining with Pattern Monotonicity
  - Constrained Mining with Data Anti-Monotonicity
  - Constrained Mining with Succinct Constraints
  - Constrained Mining with Convertible Constraints
- Handling Multiple Constraints
- Constraint-Based Sequential-Pattern Mining
Why Constraint-Based Mining?

- Finding all the patterns in a dataset autonomously?—unrealistic!
  - Too many patterns but not necessarily user-interested!

- Pattern mining in practice: Often a user-guided, interactive process
  - User directs what to be mined using a data mining query language (or a graphical user interface), specifying various kinds of constraints

- What is constraint-based mining?
  - Mine together with user-provided constraints

- Why constraint-based mining?
  - User flexibility: User provides constraints on what to be mined
  - Optimization: System explores such constraints for mining efficiency
    - E.g., Push constraints deeply into the mining process
Various Kinds of User-Specified Constraints in Data Mining

- **Knowledge type constraint**—Specifying what kinds of knowledge to mine
  - Ex.: Classification, association, clustering, outlier finding, ...

- **Data constraint**—using SQL-like queries
  - Ex.: Find products sold together in NY stores this year

- **Dimension/level constraint**—similar to projection in relational database
  - Ex.: In relevance to region, price, brand, customer category

- **Interestingness constraint**—various kinds of thresholds
  - Ex.: Strong rules: $\min_{sup} \geq 0.02$, $\min_{conf} \geq 0.6$, $\min_{correlation} \geq 0.7$

- **Rule (or pattern) constraint**
  - Ex.: Small sales (price < $10) triggers big sales (sum > $200)

The focus of this study
A constraint $c$ is **anti-monotone**
- If an itemset $S$ violates constraint $c$, so does any of its superset
- That is, mining on itemset $S$ can be terminated

- **Ex. 1:** $c_1: \text{sum}(S.\text{price}) \leq v$ is anti-monotone
- **Ex. 2:** $c_2: \text{range}(S.\text{profit}) \leq 15$ is anti-monotone
  - Itemset $ab$ violates $c_2$ ($\text{range}(ab) = 40$)
  - So does every superset of $ab$

- **Ex. 3.** $c_3: \text{sum}(S.\text{Price}) \geq v$ is **not** anti-monotone
- **Ex. 4.** Is $c_4$: $\text{support}(S) \geq \sigma$ anti-monotone?
  - Yes! Apriori pruning is essentially pruning with an anti-monotonic constraint!

---

### Note:
- Item price > 0
- Profit can be negative
Pattern Monotonicity and Its Roles

- A constraint $c$ is **monotone**: If an itemset $S$ **satisfies** the constraint $c$, so does any of its superset
  - That is, we do not need to check $c$ in subsequent mining
  - Ex. 1: $c_1$: $\text{sum}(S.Price) \geq v$ is monotone
  - Ex. 2: $c_2$: $\text{min}(S.Price) \leq v$ is monotone
  - Ex. 3: $c_3$: range($S.profit$) $\geq$ 15 is monotone
  - Itemset $ab$ satisfies $c_3$
  - So does every superset of $ab$

### Table

<table>
<thead>
<tr>
<th>TID</th>
<th>Transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>a, b, c, d, f, h</td>
</tr>
<tr>
<td>20</td>
<td>b, c, d, f, g, h</td>
</tr>
<tr>
<td>30</td>
<td>b, c, d, f, g</td>
</tr>
<tr>
<td>40</td>
<td>a, c, e, f, g</td>
</tr>
</tbody>
</table>

**min_sup = 2**

<table>
<thead>
<tr>
<th>Item</th>
<th>Price</th>
<th>Profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>100</td>
<td>40</td>
</tr>
<tr>
<td>b</td>
<td>40</td>
<td>0</td>
</tr>
<tr>
<td>c</td>
<td>150</td>
<td>-20</td>
</tr>
<tr>
<td>d</td>
<td>35</td>
<td>-15</td>
</tr>
<tr>
<td>e</td>
<td>55</td>
<td>-30</td>
</tr>
<tr>
<td>f</td>
<td>45</td>
<td>-10</td>
</tr>
<tr>
<td>g</td>
<td>80</td>
<td>20</td>
</tr>
<tr>
<td>h</td>
<td>10</td>
<td>5</td>
</tr>
</tbody>
</table>

Note: item.price > 0
Profit can be negative
Data Space Pruning with Data Anti-Monotonicity

A constraint $c$ is **data anti-monotone**: In the mining process, if a data entry $t$ cannot satisfy a pattern $p$ under $c$, $t$ cannot satisfy $p$’s superset either.

- Data space pruning: Data entry $t$ can be pruned.
- Ex. 1: $c_1: \text{sum}(S.Profit) \geq v$ is data anti-monotone.
  - Let constraint $c_1$ be: $\text{sum}(S.Profit) \geq 25$
    - $T_{30}: \{b, c, d, f, g\}$ can be removed since none of their combinations can make an $S$ whose sum of the profit is $\geq 25$
- Ex. 2: $c_2: \text{min}(S.Price) \leq v$ is data anti-monotone.
  - Consider $v = 5$ but every item in a transaction, say $T_{50}$, has a price higher than 10.
- Ex. 3: $c_3: \text{range}(S.Profit) > 25$ is data anti-monotone.

### Example Transactions

<table>
<thead>
<tr>
<th>TID</th>
<th>Transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>a, b, c, d, f, h</td>
</tr>
<tr>
<td>20</td>
<td>b, c, d, f, g, h</td>
</tr>
<tr>
<td>30</td>
<td>b, c, d, f, g</td>
</tr>
<tr>
<td>40</td>
<td>a, c, e, f, g</td>
</tr>
</tbody>
</table>

### Example Items

<table>
<thead>
<tr>
<th>Item</th>
<th>Price</th>
<th>Profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>100</td>
<td>40</td>
</tr>
<tr>
<td>b</td>
<td>40</td>
<td>0</td>
</tr>
<tr>
<td>c</td>
<td>150</td>
<td>-20</td>
</tr>
<tr>
<td>d</td>
<td>35</td>
<td>-15</td>
</tr>
<tr>
<td>e</td>
<td>55</td>
<td>-30</td>
</tr>
<tr>
<td>f</td>
<td>45</td>
<td>-10</td>
</tr>
<tr>
<td>g</td>
<td>80</td>
<td>20</td>
</tr>
<tr>
<td>h</td>
<td>10</td>
<td>5</td>
</tr>
</tbody>
</table>

**Note:** item.price > 0
Profit can be negative.
Expressing Patterns in Compressed Form: Closed Patterns

- How to handle such a challenge?

- **Solution 1: Closed patterns:** A pattern (itemset) \( X \) is closed if \( X \) is frequent, and there exists no super-pattern \( Y \supset X \), with the same support as \( X \)

  - Let Transaction DB TDB\(_1\): \( T_1: \{a_1, \ldots, a_{50}\}; \ T_2: \{a_1, \ldots, a_{100}\} \)

  - Suppose \( \text{minsup} = 1 \). How many closed patterns does TDB\(_1\) contain?
    - Two: \( P_1: \{a_1, \ldots, a_{50}\}: 2 \); \( P_2: \{a_1, \ldots, a_{100}\}: 1 \)

- Closed pattern is a lossless compression of frequent patterns
  - Reduces the # of patterns but does not lose the support information!
  - You will still be able to say: \( \{a_{2}, \ldots, a_{40}\}: 2 \), \( \{a_{5}, a_{51}\}: 1 \)
Solution 2: Max-patterns: A pattern X is a maximal frequent pattern or max-pattern if X is frequent and there exists no frequent super-pattern Y ⊇ X

Difference from close-patterns?
- Do not care the real support of the sub-patterns of a max-pattern
- Let Transaction DB TDB₁: T₁: \{a₁, ..., a₅₀\}; T₂: \{a₁, ..., a₁₀₀\}
- Suppose minsup = 1. How many max-patterns does TDB₁ contain?
  - One: P: \{a₁, ..., a₁₀₀\}: 1”

Max-pattern is a lossy compression!
- We only know \{a₁, ..., a₄₀\} is frequent
- But we do not know the real support of \{a₁, ..., a₄₀\}, ..., any more!
- Thus in many applications, close-patterns are more desirable than max-patterns
Scaling FP-growth by Item-Based Data Projection

- What if FP-tree cannot fit in memory?—Do not construct FP-tree
  - “Project” the database based on frequent single items
  - Construct & mine FP-tree for each projected DB
- Parallel projection vs. partition projection
  - Parallel projection: Project the DB on each frequent item
    - Space costly, all partitions can be processed in parallel
  - Partition projection: Partition the DB in order
    - Passing the unprocessed parts to subsequent partitions

Trans. DB

\[ f_2 \ f_3 \ f_4 \ g \ h \]

\[ f_3 \ f_4 \ i \ j \]

\[ f_2 \ f_4 \ k \]

\[ f_1 \ f_3 \ h \]

... Assume only f’s are frequent & the frequent item ordering is: \( f_1 \ f_2 \ f_3 \ f_4 \)

Parallel projection

\[ f_4\text{-proj. DB} \]

\[ f_3\text{-proj. DB} \]

\[ f_2 \ f_3 \]

\[ f_2 \]

\[ f_3 \]

Partition projection

\[ f_4\text{-proj. DB} \]

\[ f_3\text{-proj. DB} \]

\[ f_2 \ f_3 \]

\[ f_2 \]

\[ f_3 \]

\[ ... \]

\[ f_2 \text{ will be projected to } f_3\text{-proj. DB} \]

DB only when processing \( f_4\text{-proj. DB} \)
Analysis of DBLP Coauthor Relationships

- DBLP: Computer science research publication bibliographic database
  - > 3.8 million entries on authors, paper, venue, year, and other information

<table>
<thead>
<tr>
<th>ID</th>
<th>Author A</th>
<th>Author B</th>
<th>$s(A \cup B)$</th>
<th>$s(A)$</th>
<th>$s(B)$</th>
<th>Jaccard</th>
<th>Cosine</th>
<th>Kulc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Hans-Peter Kriegel</td>
<td>Martin Ester</td>
<td>28</td>
<td>146</td>
<td>54</td>
<td>0.163 (2)</td>
<td>0.315 (7)</td>
<td>0.355 (9)</td>
</tr>
<tr>
<td>2</td>
<td>Michael Carey</td>
<td>Miron Livny</td>
<td>26</td>
<td>104</td>
<td>58</td>
<td>0.191 (1)</td>
<td>0.335 (4)</td>
<td>0.349 (10)</td>
</tr>
<tr>
<td>3</td>
<td>Hans-Peter Kriegel</td>
<td>Joerg Sander</td>
<td>24</td>
<td>146</td>
<td>36</td>
<td>0.152 (3)</td>
<td>0.331 (5)</td>
<td>0.416 (8)</td>
</tr>
<tr>
<td>4</td>
<td>Christos Faloutsos</td>
<td>Spiros Papadimitriou</td>
<td>20</td>
<td>162</td>
<td>26</td>
<td>0.119 (7)</td>
<td>0.308 (10)</td>
<td>0.446 (7)</td>
</tr>
<tr>
<td>5</td>
<td>Hans-Peter Kriegel</td>
<td>Martin Pfeifle</td>
<td>18</td>
<td>146</td>
<td>18</td>
<td>0.123 (6)</td>
<td>0.351 (2)</td>
<td>0.562 (2)</td>
</tr>
<tr>
<td>6</td>
<td>Hector Garcia-Molina</td>
<td>Wilbur Labio</td>
<td>16</td>
<td>144</td>
<td>18</td>
<td>0.110 (9)</td>
<td>0.314 (8)</td>
<td>0.500 (4)</td>
</tr>
<tr>
<td>7</td>
<td>Divyakant Agrawal</td>
<td>Wang Hsiung</td>
<td>16</td>
<td>120</td>
<td>16</td>
<td>0.133 (5)</td>
<td>0.365 (1)</td>
<td>0.567 (1)</td>
</tr>
<tr>
<td>8</td>
<td>Elke Rundensteiner</td>
<td>Murali Mani</td>
<td>16</td>
<td>104</td>
<td>20</td>
<td>0.148 (4)</td>
<td>0.351 (3)</td>
<td>0.477 (6)</td>
</tr>
<tr>
<td>9</td>
<td>Divyakant Agrawal</td>
<td>Oliver Po</td>
<td>12</td>
<td>120</td>
<td>12</td>
<td>0.100 (10)</td>
<td>0.316 (6)</td>
<td>0.550 (3)</td>
</tr>
<tr>
<td>10</td>
<td>Gerhard Weikum</td>
<td>Martin Theobald</td>
<td>12</td>
<td>106</td>
<td>14</td>
<td>0.111 (8)</td>
<td>0.312 (9)</td>
<td>0.485 (5)</td>
</tr>
</tbody>
</table>

Advisor-advisee relation: Kulc: high, Jaccard: low, cosine: middle

- Which pairs of authors are strongly related?
  - Use Kulc to find Advisor-advisee, close collaborators
Analysis of DBLP Coauthor Relationships

- **DBLP**: Computer science research publication bibliographic database
- > 3.8 million entries on authors, paper, venue, year, and other information

<table>
<thead>
<tr>
<th>ID</th>
<th>Author A</th>
<th>Author B</th>
<th>$s(A \cup B)$</th>
<th>$s(A)$</th>
<th>$s(B)$</th>
<th>Jaccard</th>
<th>Cosine</th>
<th>Kulc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Hans-Peter Kriegel</td>
<td>Martin Ester</td>
<td>28</td>
<td>146</td>
<td>54</td>
<td>0.163 (2)</td>
<td>0.315 (7)</td>
<td>0.355 (9)</td>
</tr>
<tr>
<td>2</td>
<td>Michael Carey</td>
<td>Miron Livny</td>
<td>26</td>
<td>104</td>
<td>58</td>
<td>0.191 (1)</td>
<td>0.335 (4)</td>
<td>0.349 (10)</td>
</tr>
<tr>
<td>3</td>
<td>Hans-Peter Kriegel</td>
<td>Joerg Sander</td>
<td>24</td>
<td>146</td>
<td>36</td>
<td>0.152 (3)</td>
<td>0.331 (5)</td>
<td>0.416 (8)</td>
</tr>
<tr>
<td>4</td>
<td>Christos Faloutsos</td>
<td>Spiros Papadimitriou</td>
<td>20</td>
<td>162</td>
<td>26</td>
<td>0.119 (7)</td>
<td>0.308 (10)</td>
<td>0.446 (7)</td>
</tr>
<tr>
<td>5</td>
<td>Hans-Peter Kriegel</td>
<td>Martin Pfeifle</td>
<td>18</td>
<td>146</td>
<td>18</td>
<td>0.123 (6)</td>
<td>0.351 (2)</td>
<td>0.562 (2)</td>
</tr>
<tr>
<td>6</td>
<td>Hector Garcia-Molina</td>
<td>Wilbur Labio</td>
<td>16</td>
<td>144</td>
<td>18</td>
<td>0.110 (9)</td>
<td>0.314 (8)</td>
<td>0.500 (4)</td>
</tr>
<tr>
<td>7</td>
<td>Divyakant Agrawal</td>
<td>Wang Hsiung</td>
<td>16</td>
<td>120</td>
<td>16</td>
<td>0.133 (5)</td>
<td>0.365 (1)</td>
<td>0.567 (1)</td>
</tr>
<tr>
<td>8</td>
<td>Elke Rundensteiner</td>
<td>Murali Mani</td>
<td>16</td>
<td>104</td>
<td>20</td>
<td>0.148 (4)</td>
<td>0.351 (3)</td>
<td>0.477 (6)</td>
</tr>
<tr>
<td>9</td>
<td>Divyakant Agrawal</td>
<td>Oliver Po</td>
<td>12</td>
<td>120</td>
<td>12</td>
<td>0.100 (10)</td>
<td>0.316 (6)</td>
<td>0.550 (3)</td>
</tr>
<tr>
<td>10</td>
<td>Gerhard Weikum</td>
<td>Martin Theobald</td>
<td>12</td>
<td>106</td>
<td>14</td>
<td>0.111 (8)</td>
<td>0.312 (9)</td>
<td>0.485 (5)</td>
</tr>
</tbody>
</table>

Advisor-advisee relation: Kulc: high, Jaccard: low, cosine: middle

- Which pairs of authors are strongly related?
  - Use Kulc to find Advisor-advisee, close collaborators
What Measures to Choose for Effective Pattern Evaluation?

- Null value cases are predominant in many large datasets
  - Neither milk nor coffee is in most of the baskets; neither Mike nor Jim is an author in most of the papers; ......

- *Null-invariance* is an important property

- Lift, $\chi^2$ and cosine are good measures if null transactions are not predominant
  - Otherwise, Kulczynski + Imbalance Ratio should be used to judge the interestingness of a pattern

- Exercise: Mining research collaborations from research bibliographic data
  - Find a group of frequent collaborators from research bibliographic data (e.g., DBLP)
  - Can you find the likely advisor-advisee relationship and during which years such a relationship happened?

- Ref.: C. Wang, J. Han, Y. Jia, J. Tang, D. Zhang, Y. Yu, and J. Guo, "Mining Advisor-Advisee Relationships from Research Publication Networks", KDD'10
Mining Compressed Patterns

Why mining compressed patterns?
- Too many scattered patterns but not so meaningful

Pattern distance measure
\[
\text{Dist}(P_1, P_2) = 1 - \frac{|T(P_1) \cap T(P_2)|}{|T(P_1) \cup T(P_2)|}
\]

\(\delta\)-clustering: For each pattern \(P\), find all patterns which can be expressed by \(P\) and whose distance to \(P\) is within \(\delta\) (\(\delta\)-cover)

All patterns in the cluster can be represented by \(P\)

Method for efficient, direct mining of compressed frequent patterns (e.g., D. Xin, J. Han, X. Yan, H. Cheng, "On Compressing Frequent Patterns", Knowledge and Data Engineering, 60:5-29, 2007)

<table>
<thead>
<tr>
<th>Pat-ID</th>
<th>Item-Sets</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>{38,16,18,12}</td>
<td>205227</td>
</tr>
<tr>
<td>P2</td>
<td>{38,16,18,12,17}</td>
<td>205211</td>
</tr>
<tr>
<td>P3</td>
<td>{39,38,16,18,12,17}</td>
<td>101758</td>
</tr>
<tr>
<td>P4</td>
<td>{39,16,18,12,17}</td>
<td>161563</td>
</tr>
<tr>
<td>P5</td>
<td>{39,16,18,12}</td>
<td>161576</td>
</tr>
</tbody>
</table>

- Closed patterns
  - P1, P2, P3, P4, P5
  - Emphasizes too much on support
  - There is no compression

- Max-patterns
  - P3: information loss

- Desired output (a good balance):
  - P2, P3, P4
Redundancy-Aware Top-k Patterns

- Desired patterns: high significance & low redundancy

- Method: Use MMS (Maximal Marginal Significance) for measuring the combined significance of a pattern set

- Xin et al., Extracting Redundancy-Aware Top-K Patterns, KDD’06
Multi-level association mining may generate many redundant rules.

Redundancy filtering: Some rules may be redundant due to “ancestor” relationships between items.

- milk ⇒ wheat bread [support = 8%, confidence = 70%] (1)
- 2% milk ⇒ wheat bread [support = 2%, confidence = 72%] (2)

  Suppose the “2% milk” sold is about “¼” of milk sold.
  Does (2) provide any novel information?

A rule is redundant if its support is close to the “expected” value, according to its “ancestor” rule, and it has a similar confidence as its “ancestor”.

Rule (1) is an ancestor of rule (2), which one to prune?
Succinctness:

Given \( A_1 \), the set of items satisfying a succinctness constraint \( C \), then any set \( S \) satisfying \( C \) is based on \( A_1 \), i.e., \( S \) contains a subset belonging to \( A_1 \).

Idea: Without looking at the transaction database, whether an itemset \( S \) satisfies constraint \( C \) can be determined based on the selection of items.

- \( \min(S.Price) \leq v \) is succinct
- \( \sum(S.Price) \geq v \) is not succinct

Optimization: If \( C \) is succinct, \( C \) is pre-counting pushable.
## Which Constraints Are Succinct?

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Succinct</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v \in S$</td>
<td>yes</td>
</tr>
<tr>
<td>$S \supseteq V$</td>
<td>yes</td>
</tr>
<tr>
<td>$S \subseteq V$</td>
<td>yes</td>
</tr>
<tr>
<td>$\min(S) \leq v$</td>
<td>yes</td>
</tr>
<tr>
<td>$\min(S) \geq v$</td>
<td>yes</td>
</tr>
<tr>
<td>$\max(S) \leq v$</td>
<td>yes</td>
</tr>
<tr>
<td>$\max(S) \geq v$</td>
<td>yes</td>
</tr>
<tr>
<td>$ \sum(S) \leq v \ (a \in S, a \geq 0)$</td>
<td>no</td>
</tr>
<tr>
<td>$ \sum(S) \geq v \ (a \in S, a \geq 0)$</td>
<td>no</td>
</tr>
<tr>
<td>$\text{range}(S) \leq v$</td>
<td>no</td>
</tr>
<tr>
<td>$\text{range}(S) \geq v$</td>
<td>no</td>
</tr>
<tr>
<td>$\text{avg}(S) \theta v, \theta \in {=, \leq, \geq}$</td>
<td>no</td>
</tr>
<tr>
<td>$\text{support}(S) \geq \xi$</td>
<td>no</td>
</tr>
<tr>
<td>$\text{support}(S) \leq \xi$</td>
<td>no</td>
</tr>
</tbody>
</table>
Push a Succinct Constraint Deep

Database D

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1 3 4</td>
</tr>
<tr>
<td>200</td>
<td>2 3 5</td>
</tr>
<tr>
<td>300</td>
<td>1 2 3 5</td>
</tr>
<tr>
<td>400</td>
<td>2 5</td>
</tr>
</tbody>
</table>

Scan D

C₁

<table>
<thead>
<tr>
<th>itemset</th>
<th>sup.</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1}</td>
<td>2</td>
</tr>
<tr>
<td>{2}</td>
<td>3</td>
</tr>
<tr>
<td>{3}</td>
<td>3</td>
</tr>
<tr>
<td>{4}</td>
<td>1</td>
</tr>
<tr>
<td>{5}</td>
<td>3</td>
</tr>
</tbody>
</table>

L₁

<table>
<thead>
<tr>
<th>itemset</th>
<th>sup.</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1}</td>
<td>2</td>
</tr>
<tr>
<td>{2}</td>
<td>3</td>
</tr>
<tr>
<td>{3}</td>
<td>3</td>
</tr>
<tr>
<td>{5}</td>
<td>3</td>
</tr>
</tbody>
</table>

C₂

<table>
<thead>
<tr>
<th>itemset</th>
<th>sup</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1 2}</td>
<td>1</td>
</tr>
<tr>
<td>{1 3}</td>
<td>2</td>
</tr>
<tr>
<td>{1 5}</td>
<td>1</td>
</tr>
<tr>
<td>{2 3}</td>
<td>2</td>
</tr>
<tr>
<td>{2 5}</td>
<td>3</td>
</tr>
<tr>
<td>{3 5}</td>
<td>2</td>
</tr>
</tbody>
</table>

L₂

<table>
<thead>
<tr>
<th>itemset</th>
<th>sup</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1 2}</td>
<td>1</td>
</tr>
<tr>
<td>{1 3}</td>
<td>2</td>
</tr>
<tr>
<td>{1 5}</td>
<td>1</td>
</tr>
<tr>
<td>{2 3}</td>
<td>2</td>
</tr>
<tr>
<td>{2 5}</td>
<td>3</td>
</tr>
<tr>
<td>{3 5}</td>
<td>2</td>
</tr>
</tbody>
</table>

C₃

<table>
<thead>
<tr>
<th>itemset</th>
</tr>
</thead>
<tbody>
<tr>
<td>{2 3 5}</td>
</tr>
</tbody>
</table>

Scan D

C₄

itemset

<table>
<thead>
<tr>
<th>itemset</th>
</tr>
</thead>
<tbody>
<tr>
<td>{2 3 5}</td>
</tr>
</tbody>
</table>

L₃

<table>
<thead>
<tr>
<th>itemset</th>
<th>sup</th>
</tr>
</thead>
<tbody>
<tr>
<td>{2 3 5}</td>
<td>2</td>
</tr>
</tbody>
</table>

Constraint: min{S.price <= 1}
Sequential Pattern Mining

- Sequential Pattern and Sequential Pattern Mining
- GSP: Apriori-Based Sequential Pattern Mining
- SPADE: Sequential Pattern Mining in Vertical Data Format
- PrefixSpan: Sequential Pattern Mining by Pattern-Growth
- CloSpan: Mining Closed Sequential Patterns
GSP: Candidate Generation

The sequence \(< (1,2) (3) (5) >\) is dropped in the pruning phase, since its contiguous subsequence \(< (1) (3) (5) >\) is not frequent.

<table>
<thead>
<tr>
<th>Frequent 3-Sequences</th>
<th>Candidate 4-Sequences after join</th>
<th>after pruning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt; (1, 2) (3) &gt;)</td>
<td>(&lt; (1, 2) (3, 4) &gt;)</td>
<td>(&lt; (1, 2) (3, 4) &gt;)</td>
</tr>
<tr>
<td>(&lt; (1, 2) (4) &gt;)</td>
<td>(&lt; (1, 2) (3) (5) &gt;)</td>
<td></td>
</tr>
<tr>
<td>(&lt; (1) (3, 4) &gt;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(&lt; (1, 3) (5) &gt;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(&lt; (2) (3, 4) &gt;)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(&lt; (2) (3) (5) &gt;)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3: Candidate Generation: Example
GSP Algorithm: Apriori Candidate Generation

The \texttt{apriori-generate} function takes as argument $L_{k-1}$, the set of all large $(k-1)$-sequences. The function works as follows. First, join $L_{k-1}$ with $L_{k-1}$:

\begin{verbatim}
insert into $C_k$
select $p.\text{items}_{1}$, ..., $p.\text{items}_{k-1}$, $q.\text{items}_{k-1}$
from $L_{k-1} \ p$, $L_{k-1} \ q$
where $p.\text{items}_{1} = q.\text{items}_{1}$, ..., $p.\text{items}_{k-2} = q.\text{items}_{k-2}$;
\end{verbatim}

Next, delete all sequences $c \in C_k$ such that some $(k-1)$-subsequence of $c$ is not in $L_{k-1}$.

\begin{table}[h!]
\centering
\begin{tabular}{|c|c|c|}
\hline
Large 3-Sequences & Candidate 4-Sequences (after join) & Candidate 4-Sequences (after pruning) \\
\hline
\{1 2 3\} & \{1 2 3 4\} & \{1 2 3 4\} \\
\{1 2 4\} & \{1 2 4 3\} & \{1 2 3 4\} \\
\{1 3 4\} & \{1 3 4 5\} & \{1 3 5 4\} \\
\{1 3 5\} & \{1 3 5 4\} & \{1 2 3 4\} \\
\{2 3 4\} & \{1 2 4 3\} & \{1 3 5 4\} \\
\hline
\end{tabular}
\caption{Candidate Generation}
\end{table}