Data Mining
(EECS 6412)

Sequential Pattern Mining

Aijun An
Department of Electrical Engineering and Computer Science
York University

Outline

- Basic concepts of sequential pattern mining
- A Simplified Version of GSP Algorithm
- PrefixSpan

2

An Example Sequence Database

Customer Id	TransactionTime	Items Bought
1	June 25 '93	30
1	June 30 '93	90
2	June 10 '93	10, 20
2	June 15 '93	30
2	June 20 '93	40, 60, 70
3	June 25 '93	30, 50, 70
4	June 25 '93	30
4	June 30 '93	40, 70
4	July 25 '93	90
5	June 12 '93	90

Figure 1: Database Sorted by Customer Id and Transaction Time

Customer Id	Customer Sequence
1	⟨(30) (90)⟩
2	⟨(10 20) (30) (40 60 70)⟩
3	⟨(30 50 70)⟩
4	⟨(30) (40 70) (90)⟩
5	⟨(90)⟩

Figure 2: Customer-Sequence Version of the Database

- A *sequence database* consists of a set of sequences
- A *sequence* is an ordered list of itemsets
- *Itemset*: a set of items
- Items within an itemset are unordered.
- *Element*: an itemset in a sequence

3

Example of a Sequential Pattern

- Database of an online book store
  - Contains data sequences
  - Each sequence corresponds to a list of transactions by a given customer.
  - Each transaction contains the books selected by the customer in one order.
- A sequential pattern
  - 5% of customers bought "*Foundation*", then "*Foundation and Empire*" and "*Ringworld*", then "*Second Foundation*".
- Usefulness
  - Making recommendations.
  - Knowing what to stock.

4

Mining Sequential Patterns

- Objective
  - Finding (interesting) frequent sequences from a sequence database.
- Applications
  - Customer shopping sequence analysis
  - Web log analysis
  - DNA or protein analysis
  - Stock market analysis
  - Medical domain
    - Predict outset of a disease from a sequence of symptoms, etc.
  - Earthquake prediction
  - etc.

5

Basic Concepts

- A sequence $\langle a_1, a_2, \ldots, a_n \rangle$ *is contained in* (or *is a subsequence of*) another sequence $\langle b_1, b_2, \ldots, b_m \rangle$ if there exist integers $i_1 < i_2 < \ldots < i_n$ such that $a_1 \subseteq b_{i1}$, $a_2 \subseteq b_{i2}$, …, $a_n \subseteq b_{in}$.
  - E.g., ⟨(3) (4 5) (8)⟩ is contained in ⟨(7) (3 8) (9) (4 5 6) (8) ⟩
- *Support of a sequence* is defined as the fraction of total sequences in the database that contains this sequence.
  - Support of ⟨(30) (90)⟩?
  - Support of ⟨(20) (60 70)⟩?
  - Support of <(30)(70)>?

Customer Id	Customer Sequence
1	⟨(30) (90)⟩
2	⟨(10 20) (30) (40 60 70)⟩
3	⟨(30 50 70)⟩
4	⟨(30) (40 70) (90)⟩
5	⟨(90)⟩

- *Frequent sequences* (also called *sequential patterns*)
  - sequences that satisfy a minimum support (min_sup).

6

Sequential Pattern Mining

- What is sequential pattern mining
  - Given a sequence database, find the set of frequent sequences that satisfy a minimum support (min_sup).
- Algorithms
  - Initial Apriori-like algorithms (Agrawal and Srikant, 95)
    - AprioriAll, AprioriSome, and DynamicSome
  - GSP - an Apriori-like, influential mining method (Srikant and Agrawal, 96)
  - PrefixSpan (Pei, et al, 01)
  - SPADE (Zaki, 01)
  - Mining sequential patterns with constraints (Pei, et al, 02)
  - etc.

7

Apriori Property

- The Apriori property in sequential patterns:

  <u>Any nonempty subsequence of a frequent sequence must be frequent</u>

  - If a sequence is infrequent, then none of its super-sequences is frequent.
  - i.e., if ⟨(3) (4 5)⟩ is infrequent, so are ⟨(3) (4 5) (8)⟩ and ⟨(3 6) (4 5)⟩

8

GSP

- GSP: Generalized Sequential Pattern Mining
- Proposed in
  - R. Srikant and R. Agrawal. Mining Sequential Patterns: Generalizations and Performance Improvements. In Proc. of EDBT'96.
  - Can be downloaded from the course web site
- GSP considers some time constraints and item taxonomy
  - More general than simply mining sequential patterns

9

A Simplified Version of GSP

- GSP without the generalizations
- Problem statement:
  - find all frequent sequences from a database of sequences given a *min_sup*
- Length of a sequence = number of items in the sequence
  - Length of <(a)(b)> is 2
  - Length of <(a b)> is 2
  - Length of <(a b) (c)> is 3
- A *length-k sequence* is also called *k-sequence*.

10

General Description of Simplified GSP

- Method
  - Take sequences in form of <(x)> as length-1 candidates
  - Scan database once, find L_1, the set of length-1 sequential patterns
  - Let k=1; while L_k is not empty do
    - Form C_{k+1}, the set of length-(k+1) candidates from L_k;
    - If C_{k+1} is not empty, scan database once, find L_{k+1}, the set of length-(k+1) sequential patterns
    - Let k=k+1;

Finding Length-1 Sequential Patterns

- Initial candidates
  - <(a)>, <(b)>, <(c)>, <(d)>, <(e)>, <(f)>, <(g)>, <(h)>
- Scan database once
  - count support for candidates

min_sup =40%

min_sup_count =2

Seq-id	Sequence
10	<(bd)(c)(b)(ac)>
20	<(bf)(ce)(b)(fg)>
30	<(ah)(bf)(a)(b)(f)>
40	<(be)(ce)(d)>
50	<(a)(bd)(b)(c)(b)(ade)>

Cand	Sup
<(a)>	**3**
<(b)>	**5**
<(c)>	**4**
<(d)>	**3**
<(e)>	**3**
<(f)>	**2**
~~<(g)>~~	1
~~<(h)>~~	1

Length-2 Candidate Generation

- How to generate C_2 from L_1
  - Merge every pair of frequent length-1 sequences.
  - Merging two frequent length-1 sequences $<(i_1)>$ and $<(i_2)>$ will produce three candidate 2-sequences:

    $$<(i_1)\ (i_2)> \quad <(i_2)\ (i_1)> \text{ and } <(i_1\ i_2)>$$

    assuming that i_1 is different from i_2.
  - Every frequent length-1 sequence <(i)> can join with itself to produce <(i)(i)>.

13

Length-2 Candidates
(Cont'd from the example on Slide 13)

Length of a sequence= the number of items

51 length-2 Candidates

	<a>	<b>	<c>	<d>	<e>	<f>
<a>	<(a)(a)>	<(a)(b)>	<(a)(c)>	<(a)(d)>	<(a)(e)>	<(a)(f)>
<b>	<(b)(a)>	<(b)(b)>	<(b)(c)>	<(b)(d)>	<(b)(e)>	<(b)(f)>
<c>	<(c)(a)>	<(c)(b)>	<(c)(c)>	<(c)(d)>	<(c)(e)>	<(c)(f)>
<d>	<(d)(a)>	<(d)(b)>	<(d)(c)>	<(d)(d)>	<(d)(e)>	<(d)(f)>
<e>	<(e)(a)>	<(e)(b)>	<(e)(c)>	<(e)(d)>	<(e)(e)>	<(e)(f)>
<f>	<(f)(a)>	<(f)(b)>	<(f)(c)>	<(f)(d)>	<(f)(e)>	<(f)(f)>

	<a>	<b>	<c>	<d>	<e>	<f>
<a>		<(ab)>	<(ac)>	<(ad)>	<(ae)>	<(af)>
<b>			<(bc)>	<(bd)>	<(be)>	<(bf)>
<c>				<(cd)>	<(ce)>	<(cf)>
<d>					<(de)>	<(df)>
<e>						<(ef)>
<f>						

Without Apriori property, $8*8+8*7/2=92$ candidates

Apriori prunes 44.57% candidates

14

Finding Length-2 Sequential Patterns

- Scan database one more time, collect support count for each length-2 candidate
- There are 19 length-2 candidates which pass the minimum support threshold in our example DB
  - They are length-2 sequential patterns:

<(a)(a)>, <(a)(b)>
<(b)(a)>,<(b)(b)>,<(b)(c)>,<(b)(d)>,<(b)(e)>,<(b)(f)>
<(c)(a)>,<(c)(b)>,<(c)(d)>
<(d)(a)>,<(d)(b)>,<(d)(c)>
<(f)(b)>,<(f)(f)>
<(bd)>,<(bf)>,<(ce)>

15

Generating C_k from L_{k-1} (k>2)

- Join step: generate C_k by joining L_{k-1} with L_{k-1}
  - Rank all items in an itemset in an order.
  - A sequence s_1 joins with s_2 if the subsequence obtained by dropping the first item of s1 is the same as the subsequence obtained by dropping the last item of s_2.
  - The joined sequence is s_1 plus the last item of s_2.
    - The added item becomes a separate element if it was a separate element in s_2, or part of the last element of s_1 otherwise.
  - Examples:
    - Joining <(1)(2 3)(4)> and <(2 3)(4)(5)> produces <(1) (2 3)(4)(5)>
    - Joining <(1)(2 3)(4)> and <(2 3)(4 5)> produces <(1)(2 3)(4 5)>

16

Generating C_k from L_{k-1} (k>2) (Cont'd)

- Prune step: delete candidates in C_k that have infrequent (k-1)-subsequence
  - A (k-1)-subsequence of sequence *s* is derived by dropping an item from *s*
  - Check each (k-1)-subsequence against L_{k-1}
  - Example:
    - If <(ab)(d)>, <(b)(ad)>, <(b)(de)> are all length-3 frequent sequences, then
    - Join result: <(ab)(de)>
    - Its length-3 subsequences:
      - <(b)(de)>, <(a)(de)>, <(ab)(e)>, <(ab)(d)>
    - <(a)(de)> and <(ab)(e)> are infrequent, <(ab)(de)> is pruned
    - C4 becomes empty

17

Generating C_k from L_{k-1} (k>2) (Cont'd)

- More Examples:
  - If <(a)(b)>, <(a)(a)> and <(b)(a)> are all length-2 sequential patterns, then length-3 candidates are:
    - Join result:
      - <(a)(b)(a)> <(a)(a)(b)>, <(a)(a)(a)>, <(b)(a)(b)>, and <(b)(a)(a)>.
    - After pruning:
      - <(a)(b)(a)>, <(a)(a)(b)>, <(a)(a)(a)>, <(b)(a)(a)>.
  - If <(bd)>, <(b)(b)> and <(d)(b)> are all length-2 sequential patterns, what are the length-3 candidates?
    - Join result:

      <(bd)b>, <(b)(bd)>, <(b)(b)(b)>, <(d)(bd)>, <(d)(b)(b)>
    - After pruning:
      - <(bd)b>, <(b)(b)(b)>, <(d)(b)(b)>

18

Example Continued

- L_4: Length-4 sequential patterns: *min_sup_count*=2
  - <(b)(c)(b)(a)>
  - <(bd)(b)(a)>
  - <(bd)(b)(c)>
  - <(bd)(c)(a)>
  - <(bd)(c)(b)>
  - <(bf)(b)(f)>
  - <(d)(c)(b)(a)>

Seq-id	Sequence
10	<(bd)(c)(b)(ac)>
20	<(bf)(ce)(b)(fg)>
30	<(ah)(bf)(a)(b)(f)>
40	<(be)(ce)(d)>
50	<(a)(bd)(b)(c)(b)(ade)>

- C_5: Length-5 candidates (after join and prune):
  - <(bd)(c)(b)(a)>
- L_5: Length-5 sequential pattern (found by scanning DB):
  - <(bd)(c)(b)(a)>: 2

19

Summary of Simplified GSP

- Make the first pass over the sequence database D to find all the 1-element (length-1) frequent sequences
- Repeat until no new candidate or frequent sequences are found
  - **Candidate Generation**:
    - Merge joinable pairs of frequent sequences of length (k-1) to generate candidate sequences that contain k items
    - Prune candidate *k*-sequences that contain infrequent (*k-1*)-subsequences
  - **Support Counting and Candidate Elimination**:
    - Make a new pass over the sequence database D to find the support count for each candidate sequence
    - Eliminate candidate *k*-sequences whose actual support is less than *min_sup*

20

Bottlenecks of GSP

- A huge set of candidates
  - 1,000 frequent length-1 sequences generate $1000 \times 1000 + \frac{1000 \times 999}{2} = 1{,}499{,}500$ length-2 candidates!
- 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!

$$\sum_{i=1}^{100} \binom{100}{i} = 2^{100} - 1 \approx 10^{30}$$

21

Better Method: PrefixSpan

- Generate all frequent sequences <u>without candidate generation and testing</u>.
- J. Pei et al. PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. In Proc. of ICDE'01.
- Will talk about it next.

22

PrefixSpan

- Generate all frequent sequences <u>without candidate generation and testing</u>.
- Strategy
  - Divide and conquer
    - Divide the patterns to be mined into subsets and find patterns in each subset recursively.
  - Projection-based
    - Recursively project a sequence database into a set of smaller databases based on the frequent "prefix" mined so far
    - Mine each projected database to find frequent "suffixes"

Prefix and Suffix

- Suppose all the items in an element (i.e. an itemset) of a sequence are listed alphabetically.
- *Prefixes* of sequence <a(abc)(ac)d(cf)> are <a>, <aa>, <a(ab)> , <a(abc)> , <a(abc)a>, … .
- *Suffixes* of sequence <a(abc)(ac)d(cf)> :
  - <(abc)(ac)d(cf)> is the suffix wrt prefix <a>
  - <(_bc)(ac)d(cf)> is the suffix wrt prefix <aa>
  - <(_c)(ac)d(cf)> is the suffix wrt prefix <a(ab)>
  - …

Mining Sequential Patterns by Prefix Projections

- Step 1: find length-1 sequential patterns
  - <a>, <b>, <c>, <d>, <e>, <f>
- Step 2: divide search space. The complete set of freq. seq. can be partitioned into 6 subsets:
  - The ones having prefix <a>;
  - The ones having prefix <b>;
  - …
  - The ones having prefix <f>

SID	sequence
10	<a(abc)(ac)d(cf)>
20	<(ad)c(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

min_sup count=2

25

Finding Freq. Seq. with Prefix <a>

- Only need to consider sequences containing <a>
- In a sequence containing <a>, only the subsequence (suffixes) prefixed with the first occurrence of <a> should be considered.
- The collection of such subsequences is called *<a>-projected database*:

  <(abc)(ac)d(cf)>,
  <(_d)c(bc)(ae)>,
  <(_b)(df)cb>,
  <(_f)cbc>.

SID	sequence
10	<a(abc)(ac)d(cf)>
20	<(ad)c(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

min_sup count=2

Next: recursively mine *<a>-projected database* to find frequent sequences in the projected database.

26

Finding Freq. Seq. with Prefix <a> (Cont’d)

- Find local frequent length-1 sequences by scanning *<a>-projected database*
  - *<a>-projected database*: <(abc)(ac)d(cf)>, <(_d)c(bc)(ae)>, <(_b)(df)cb>, <(_f)cbc>
  - *Local frequent length-1 sequences*:
    <a>:2, <b>:4, <(_b)>:2, <c>:4, <d>:2, <f>:2

min_sup count=2

- Generate all the length-2 freq. seq. having prefix <a>: <aa>:2, <ab>:4, <(ab)>:2, <ac>:4, <ad>:2, <af>:2
- Further partition frequent sequences with prefix <a> into 6 subsets
  - Having prefix <aa>;
  - Having prefix <ab>;
  - …
  - Having prefix <af>

27

Finding Freq. Seq. with Prefix <aa>

- From <a>-projected database:
  <(abc)(ac)d(cf)>, <(_d)c(bc)(ae)>, <(_b)(df)cb>, <(_f)cbc>
- Generate <aa>-projected database:
  <(_bc))(ac)d(cf)>, <(_e)>
- No local frequent items, stop growing prefix <aa>.

min_sup count=2

SID	sequence
10	<a(abc)(ac)d(cf)>
20	<(ad)c(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

28

Finding Freq. Seq. with Prefix <ab>

- From <a>-projected database:
  <(abc)(ac)d(cf)>, <(_d)c(bc)(ae)>, <(_b)(df)cb>, <(_f)cbc>
- Generate <ab>-projected database:
  <(_c)(ac)d(cf)>, <(_c)(ae)>, <c>
- Local frequent length-1 sequences
  - <a>:2, <c>:2, <(_c)>:2
- Generate length-3 freq. seq. with prefix <ab>:
  - <aba>:2, <abc>:2, <a(bc)>:2
- Further partition the set of feq. Seq. prefixed with <ab>:
  - Sequences with prefix <aba>
  - Sequences with prefix <abc>
  - Sequences with prefix <a(bc)>

29

Finding Freq. Seq. with Prefix <aba>

- From <ab>-projected database:
  <(_c) (ac)d(cf)>, <(_c)(ae)>, <c>
- Generate <aba>-projected database:
  - <(_c)d(cf)>, <(_e)>
- No local frequent length-1 sequence, stop growing <aba>.

30

Finding Freq. Seq. with Prefix <abc>

- From <ab>-projected database:
  <(_c) (ac)d(cf)>, <(_c)(ae)>, <c>
- Generate <abc>-projected database:
  - <d(cf)>
- No local frequent length-1 sequence, stop growing <abc>.

31

Finding Freq. Seq. with Prefix <a(bc)>

- From <ab>-projected database:
  <(_c) (ac)d(cf)>, <(_c)(ae)>, <c>
- Generate <a(bc)>-projected database:
  - <(ac)d(cf)>, <(ae)>
- Local frequent length-1 sequences:
  - <a>:2
- Generate length-4 freq. seq. with prefix <a(bc)>:
  - <a(bc)a>:2
- Further mining <a(bc)a>-projected database returns no frequent sequence prefixed with <a(bc)a>.
- This finishes generating all the patterns prefixed with <ab>.

32

Completeness of PrefixSpan

SDB

SID	sequence
10	<a(abc)(ac)d(cf)>
20	<(ad)c(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

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

Having prefix <a>

Having prefix <b>

Having prefix <c>, ..., <f>

<u><a>-projected database</u>
<(abc)(ac)d(cf)>
<(_d)c(bc)(ae)>
<(_b)(df)cb>
<(_f)cbc>

Length-2 sequential
Patterns starting with a:
<aa>, <ab>, <(ab)>,
<ac>, <ad>, <af>

<u><b>-projected database</u>

…

… …

Having prefix <aa>

Having prefix <af>

<u><aa>-proj. db</u> … <u><af>-proj. db</u>

Efficiency of PrefixSpan

- No candidate sequence needs to be generated
- Projected databases keep shrinking
- Major cost of PrefixSpan:
  - constructing projected databases.
  - can be improved by *Pseudo-projections*

Speed-up by Pseudo-projection

- Major cost of PrefixSpan: projection
  - Suffixes of a sequence often appear repeatedly in recursively projected databases
    - <(abc)(ac)d(cf)> appears in <a>-projected database
    - <(_c)(ac)d(cf)> appears in <ab>-projected database
- When (projected) database can be held in main memory, use pointers to form projections
  - Pointer to the sequence
  - Offset of the suffix

s=<a(abc)(ac)d(cf)>

<a>

s|<a>: (, 2) <(abc)(ac)d(cf)>

<ab>

s|<ab>: (, 4) <(_c)(ac)d(cf)>

Pseudo-Projection vs. Physical Projection

- Pseudo-projection avoids physically copying suffixes
  - Efficient in running time and space when database can be held in main memory
- However, it is not efficient when database cannot fit in main memory
  - Disk-based random accessing is very costly
- Suggested Approach:
  - Integration of physical and pseudo-projection
  - Swapping to pseudo-projection when the projected database fits in memory

Experiments and Performance Analysis

- Comparing PrefixSpan with GSP, FreeSpan and SPADE in large databases
  - GSP (IBM Almaden, Srikant & Agrawal EDBT'96)
  - FreeSpan (J. Han J. Pei, B. Mortazavi-Asi, Q. Chen, U. Dayal, M.C. Hsu, KDD'00)
  - SPADE (Zaki, 01)
- PrefixSpan is fastest and scalable.

37

Problem of Sequential Pattern Mining

- GSP and PrefixSpan finds all the sequences that satisfy the support threshold.
- A long frequent sequence contains a combinatorial number of frequent subsequences:
  - For a length-100 sequential pattern, there exist $2^{100}-1$ nonempty subsequences.
- Problem: too many patterns are generated.

38

Solutions

- Mining maximal or closed sequential patterns
  - Maximal sequential pattern:
    - A frequent sequence *s* is *maximal* if there exists no frequent super-sequence of *s*.
  - Closed sequential pattern:
    - A sequence *s* is *closed* if there exists no super-sequence of s with the same support as *s*.
    - *CloSpan* (Yan, Han and Afshar, 2003): an efficient closed sequential pattern mining method.

39

Solutions (Cont'd)

- Constraint-based mining of sequential patterns
  - Constraints
    - Item constraint
      - Find patterns containing a, b, c, but no d.
    - Length constraint
      - Find patterns having at least (or at most) 10 items
    - Super-pattern constraint
      - Find patterns that contain <(PC)(Digital-camera)>
    - Aggregate constraint
      - Find patterns the average price of whose items is over $100.

40

Solutions (Cont'd)

- More constraints
  - Regular expression constraint
    - <a*{bb|(bc)d|dd}>
  - Duration constraint
    - Find patterns of events about +/- 24 hours of a shooting
  - Gap constraint
    - Find purchasing patterns such that the gap between each consecutive purchases is less than 1 month.

41

Solutions (Cont'd)

- Properties of constraints
  - Anti-monotonic constraint
    - If a sequence *s* satisfies constraint *C*, so does every non-empty subsequence of *s*
    - Examples: support(s) >=5%, length(s)<10
  - Monotonic constraint
    - If a sequence *s* satisfies constraint *C*, so does every super sequence of *s*.
    - Examples: length(s)>=10, super pattern constraints.
- A systematic study on constraint-based sequential pattern ming:
  - J. Pei, J. Han, and W. Wang. "Mining Sequential Patterns with Constraints in Large Databases“. *In Proceedings of CIKM'02.*

42

More Recent Research on Frequent Pattern Mining

- Mining other kinds of frequent patterns
  - Frequent tree/graph mining
    - Find common structural components
  - Examples of applications
    - XML documents can be modeled as trees
    - Molecule or biochemical structures can be modeled as graphs
    - Web connection structures can be modeled as graphs
- Finding frequent patterns from data streams
  - Only one scan of DB is allowed.

More Recent Research on Frequent Pattern Mining (Cont'd)

- Finding high-utility patterns (either itemsets or sequences)
  - Consider the quantity of the item inside a transaction.
  - Consider the importance (e.g., price) of an item
  - Find patterns whose revenue is at least, e.g., $1000

Frequent Pattern Mining Resources

- Web site: http://fimi.cs.helsinki.fi/ contains some frequent itemset mining implementations, datasets and papers.

- SPMF: open-source data mining library containing frequent sequence and itemset mining:

  http://www.philippe-fournier-viger.com/spmf/

45

Next Class

- Decision tree learning (Sections 8.1 and 8.2 in Chapter 8)

46