

# Sequential Pattern Mining

---

# Outline

---

- What is sequence database and sequential pattern mining
- Methods for sequential pattern mining
- Constraint-based sequential pattern mining
- Periodicity analysis for sequence data

# Sequence Databases

- A sequence database consists of **ordered elements** or events
- Transaction databases vs. sequence databases

## A transaction database

TID	itemsets
10	a, b, d
20	a, c, d
30	a, d, e
40	b, e, f

## A sequence database

SID	sequences
10	<a( <u>ab</u> c)(a <u>c</u> )d(cf)>
20	<(ad)c(bc)(ae)>
30	<(ef)( <u>ab</u> )(df) <u>c</u> b>
40	<eg(af)cbc>

# Applications

---

- Applications of sequential pattern mining
  - Customer shopping sequences:
    - First buy computer, then CD-ROM, and then digital camera, within 3 months.
  - Medical treatments, natural disasters (e.g., earthquakes), science & eng. processes, stocks and markets, etc.
  - Telephone calling patterns, Weblog click streams
  - DNA sequences and gene structures

# Subsequence vs. super sequence

- A sequence is an ordered list of events, denoted  $\langle e_1 e_2 \dots e_l \rangle$
- Given two sequences  $\alpha = \langle a_1 a_2 \dots a_n \rangle$  and  $\beta = \langle b_1 b_2 \dots b_m \rangle$
- $\alpha$  is called a subsequence of  $\beta$ , denoted as  $\alpha \subseteq \beta$ , if there exist integers  $1 \leq j_1 < j_2 < \dots < j_n \leq m$  such that  $a_1 \subseteq b_{j_1}, a_2 \subseteq b_{j_2}, \dots, a_n \subseteq b_{j_n}$
- $\beta$  is a super sequence of  $\alpha$ 
  - E.g.  $\alpha = \langle (ab), d \rangle$  and  $\beta = \langle (abc), (de) \rangle$

# What Is Sequential Pattern Mining?

- Given a set of sequences and support threshold, find the complete set of *frequent* subsequences

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

A sequence database

SID	sequence
10	<a( <u>ab</u> c)(a <u>c</u> )d(cf)>
20	<(ad)c(bc)(ae)>
30	<(ef)( <u>ab</u> )(df) <u>c</u> b>
40	<eg(af)cbc>

An element may contain a set of items  
Items within an element are unordered  
and we list them alphabetically.

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

Given support threshold  $min\_sup = 2$ , <(ab)c> is  
a sequential pattern

# Challenges on Sequential Pattern Mining

---

- A **huge number** of possible sequential **patterns** are hidden in databases
- A mining algorithm should
  - find the **complete set of patterns**, when possible, satisfying the minimum support (frequency) threshold
  - be highly **efficient, scalable**, involving only a small number of database scans
  - be able to incorporate various kinds of **user-specific constraints**

# Studies on Sequential Pattern Mining

---

- Concept introduction and an initial Apriori-like algorithm
  - Agrawal & Srikant. Mining sequential patterns, [ICDE'95]
- Apriori-based method: **GSP** (Generalized Sequential Patterns: Srikant & Agrawal [EDBT'96])
- Pattern-growth methods: FreeSpan & **PrefixSpan** (Han et al. KDD'00; Pei, et al. [ICDE'01])
- Vertical format-based mining: **SPADE** (Zaki [Machine Learning'00])
- Constraint-based sequential pattern mining (SPIRIT: Garofalakis, Rastogi, Shim [VLDB'99]; Pei, Han, Wang [CIKM'02])
- Mining closed sequential patterns: **CloSpan** (Yan, Han & Afshar [SDM'03])



# Methods for sequential pattern mining

---

- **Apriori-based Approaches**
  - **GSP**
  - **SPADE**
- **Pattern-Growth-based Approaches**
  - **FreeSpan**
  - **PrefixSpan**

# The Apriori Property of Sequential Patterns

---

- A basic property: Apriori (Agrawal & Srikant'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  $\rightarrow$  so do  $\langle hab \rangle$  and  $\langle (ah)b \rangle$

Seq. ID	Sequence
10	$\langle (bd)cb(ac) \rangle$
20	$\langle (bf)(ce)b(fg) \rangle$
30	$\langle (ah)(bf)abf \rangle$
40	$\langle (be)(ce)d \rangle$
50	$\langle a(bd)bcb(ade) \rangle$

Given support threshold  
 $min\_sup = 2$

# GSP—Generalized Sequential Pattern Mining

---

- GSP (Generalized Sequential Pattern) mining algorithm
- Outline of the method
  - Initially, every item in DB is a candidate of length-1
  - for each level (i.e., sequences of length- $k$ ) do
    - scan database to collect support count for each candidate sequence
    - generate candidate length- $(k+1)$  sequences from length- $k$  frequent sequences using Apriori
  - repeat until no frequent sequence or no candidate can be found
- Major strength: Candidate pruning by Apriori

# Finding Length-1 Sequential Patterns

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

*min\_sup=2*

Seq. ID	Sequence
10	<(bd)cb(ac)>
20	<(bf)(ce)b(fg)>
30	<(ah)(bf)abf>
40	<(be)(ce)d>
50	<a(bd)bcb(ade)>

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

# Generating Length-2 Candidates

51 length-2  
Candidates

	<a>	<b>	<c>	<d>	<e>	<f>
<a>	<aa>	<ab>	<ac>	<ad>	<ae>	<af>
<b>	<ba>	<bb>	<bc>	<bd>	<be>	<bf>
<c>	<ca>	<cb>	<cc>	<cd>	<ce>	<cf>
<d>	<da>	<db>	<dc>	<dd>	<de>	<df>
<e>	<ea>	<eb>	<ec>	<ed>	<ee>	<ef>
<f>	<fa>	<fb>	<fc>	<fd>	<fe>	<ff>

	<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)>

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

Apriori prunes  
 44.57% candidates

# 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
  - They are length-2 sequential patterns

# The GSP Mining Process

5<sup>th</sup> scan: 1 cand. 1 length-5 <(bd)cba>  
seq. pat.

Cand. cannot  
pass sup.  
threshold

4<sup>th</sup> scan: 8 cand. 6 length-4 <abba> <(bd)bc> ...  
seq. pat.

Cand. not in DB  
at all

3<sup>rd</sup> scan: 46 cand. 19  
length-3 seq. pat. 20 cand.  
not in DB at all

<abb> <aab> <aba> <bba> <bab> ...

2<sup>nd</sup> scan: 51 cand. 19

length-2 seq. pat. 10 cand.  
not in DB at all

<aa> <ab> ... <af> <ba> <bb> ... <ff> <(ab)> ... <(ef)>

1<sup>st</sup> scan: 8 cand. 6 length-1  
seq. pat.

<a> <b> <c> <d> <e> <f> <g> <h>

$min\_sup$   
=2

Seq. ID	Sequence
10	<(bd)cb(ac)>
20	<(bf)(ce)b(fg)>
30	<(ah)(bf)abf>
40	<(be)(ce)d>
50	<a(bd)bcb(ade)>

# The GSP Algorithm

---

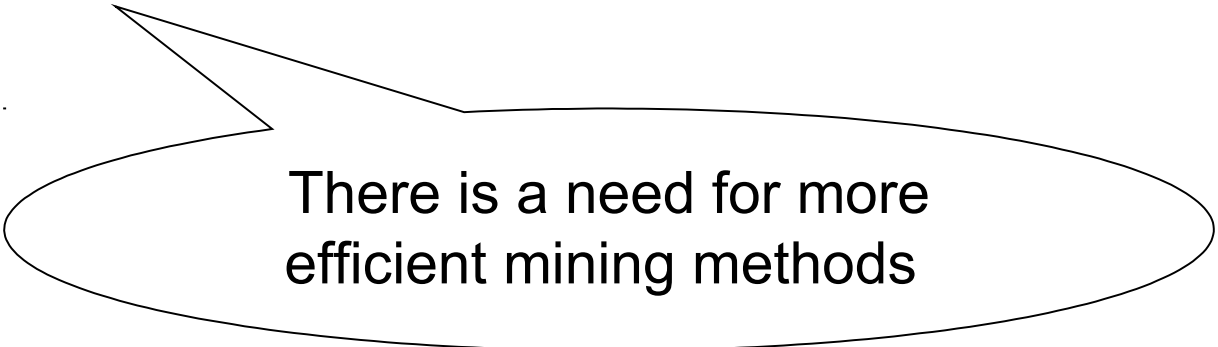
- Take sequences in form of  $\langle x \rangle$  as length-1 candidates
- Scan database once, find  $F_1$ , the set of length-1 sequential patterns
- Let  $k=1$ ; while  $F_k$  is not empty do
  - Form  $C_{k+1}$ , the set of length- $(k+1)$  candidates from  $F_k$ ;
  - If  $C_{k+1}$  is not empty, scan database once, find  $F_{k+1}$ , the set of length- $(k+1)$  sequential patterns
  - Let  $k=k+1$ ;



# The GSP Algorithm

---

- Benefits from the Apriori pruning
  - Reduces search space
- Bottlenecks
  - Scans the database multiple times
  - Generates a huge set of candidate sequences



There is a need for more efficient mining methods

# The SPADE Algorithm

---

- **SPADE** (Sequential Pattern Discovery using Equivalent Class) developed by Zaki 2001
- A vertical format sequential pattern mining method
- A sequence database is mapped to a large set of items: **<SID, EID>**
- Sequential pattern mining is performed by
  - growing the subsequences (patterns) one item at a time  
by Apriori candidate generation

# The SPADE Algorithm

SID	EID	Items
1	1	a
1	2	abc
1	3	ac
1	4	d
1	5	cf
2	1	ad
2	2	c
2	3	bc
2	4	ae
3	1	ef
3	2	ab
3	3	df
3	4	c
3	5	b
4	1	e
4	2	g
4	3	af
4	4	c
4	5	b
4	6	c

a		b		...
SID	EID	SID	EID	...
1	1	1	2	
1	2	2	3	
1	3	3	2	
2	1	3	5	
2	4	4	5	
3	2			
4	3			

ab			ba			...
SID	EID (a)	EID(b)	SID	EID (b)	EID(a)	...
1	1	2	1	2	3	
2	1	3	2	3	4	
3	2	5				
4	3	5				

aba				...
SID	EID (a)	EID(b)	EID(a)	...
1	1	2	3	
2	1	3	4	

# Bottlenecks of Candidate Generate-and-test

---

- A huge set of candidates generated.
  - Especially 2-item candidate sequence.
- Multiple Scans of database in mining.
  - The length of each candidate grows by one at each database scan.
- Inefficient for mining long sequential patterns.
  - A long pattern grows up from short patterns
  - An exponential number of short candidates

# PrefixSpan (Prefix-Projected Sequential Pattern Growth)

---

- PrefixSpan
  - Projection-based
  - But only prefix-based projection: less projections and quickly shrinking sequences
- J.Pei, J.Han,... PrefixSpan : Mining sequential patterns efficiently by prefix-projected pattern growth. ICDE'01.

# Prefix and Suffix (Projection)

- $\langle a \rangle$ ,  $\langle aa \rangle$ ,  $\langle a(ab) \rangle$  and  $\langle a(abc) \rangle$  are prefixes of sequence  $\langle a(abc)(ac)d(cf) \rangle$
- Given sequence  $\langle a(abc)(ac)d(cf) \rangle$

Prefix	<u>Suffix</u> (Prefix-Based Projection)
$\langle a \rangle$	$\langle (abc)(ac)d(cf) \rangle$
$\langle aa \rangle$	$\langle (\_bc)(ac)d(cf) \rangle$
$\langle ab \rangle$	$\langle (\_c)(ac)d(cf) \rangle$

# Mining Sequential Patterns by Prefix Projections

- Step 1: find length-1 sequential patterns
  - $\langle a \rangle$ ,  $\langle b \rangle$ ,  $\langle c \rangle$ ,  $\langle d \rangle$ ,  $\langle e \rangle$ ,  $\langle f \rangle$
- Step 2: divide search space. The complete set of seq. pat. can be partitioned into 6 subsets:
  - The ones having prefix  $\langle a \rangle$ ;
  - The ones having prefix  $\langle b \rangle$ ;
  - ...
  - The ones having prefix  $\langle f \rangle$

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

# Finding Seq. Patterns with Prefix <a>

- Only need to consider projections w.r.t. <a>
  - <a>-projected database: <(abc)(ac)d(cf)>, <(\_d)c(bc)(ae)>, <(\_b)(df)cb>, <(\_f)cbc>
- Find all the length-2 seq. pat. having prefix <a>:  
<aa>, <ab>, <(ab)>, <ac>, <ad>, <af>
  - Further partition into 6 subsets
    - Having prefix <aa>;
    - ...
    - Having prefix <af>

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



# 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>

<a>-projected database

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

<b>-projected database

...

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

Having prefix <aa>

Having prefix <af>

<aa>-proj. db

...

<af>-proj. db

# The Algorithm of PrefixSpan

---

- **Input:** A sequence database  $S$ , and the minimum support threshold  $\text{min\_sup}$
- **Output:** The complete set of sequential patterns
- **Method:** Call  $\text{PrefixSpan}(<>, 0, S)$
- **Subroutine**  $\text{PrefixSpan}(\alpha, l, S|\alpha)$
- **Parameters:**
  - $\alpha$ : sequential pattern,
  - $l$ : the length of  $\alpha$ ;
  - $S|\alpha$ : the  $\alpha$ -projected database, if  $\alpha \neq <>$ ; otherwise; the sequence database  $S$

# The Algorithm of PrefixSpan(2)

---

- **Method**

1. Scan  $S|\alpha$  once, find the set of frequent items  $b$  such that:
  - a)  $b$  can be assembled to the last element of  $\alpha$  to form a sequential pattern; or
  - b)  $\langle b \rangle$  can be appended to  $\alpha$  to form a sequential pattern.
2. For each frequent item  $b$ , append it to  $\alpha$  to form a sequential pattern  $\alpha'$ , and output  $\alpha'$ ;
3. For each  $\alpha'$ , construct  $\alpha'$ -projected database  $S|\alpha'$ , and call PrefixSpan( $\alpha'$ ,  $l+1$ ,  $S|\alpha'$ ).

# 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 bi-level projections

# Optimization in PrefixSpan

---

- **Single level vs. bi-level projection**
  - Bi-level projection with 3-way checking may reduce the number and size of projected databases
- **Physical projection vs. pseudo-projection**
  - Pseudo-projection may reduce the effort of projection when the projected database fits in main memory
- **Parallel projection vs. partition projection**
  - Partition projection may avoid the blowup of disk space

# Scaling Up by Bi-Level Projection

---

- Partition search space based on length-2 sequential patterns
- Only form projected databases and pursue recursive mining over bi-level projected databases

# Speed-up by Pseudo-projection

- Major cost of PrefixSpan: projection
  - Postfixes of sequences often appear repeatedly in recursive projected databases
- When (projected) database can be held in main memory, use **pointers** to form projections

- Pointer to the sequence
- Offset of the postfix

$s = \langle a(abc)(ac)d(cf) \rangle$

$s| \langle a \rangle : ( , 2) \langle (abc)(ac)d(cf) \rangle$

$s| \langle ab \rangle : ( , 4) \langle (_c)(ac)d(cf) \rangle$

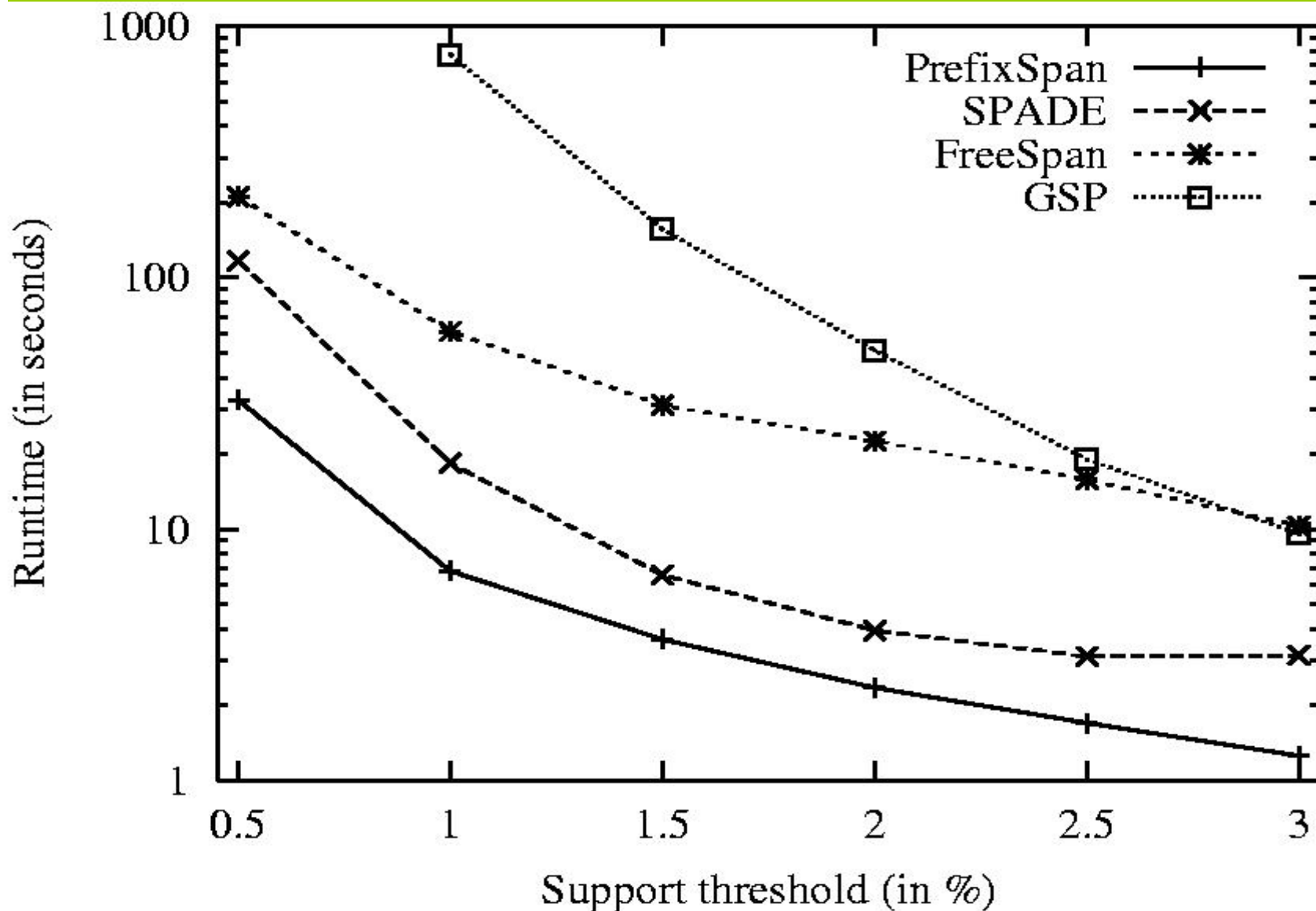
# Pseudo-Projection vs. Physical Projection

---

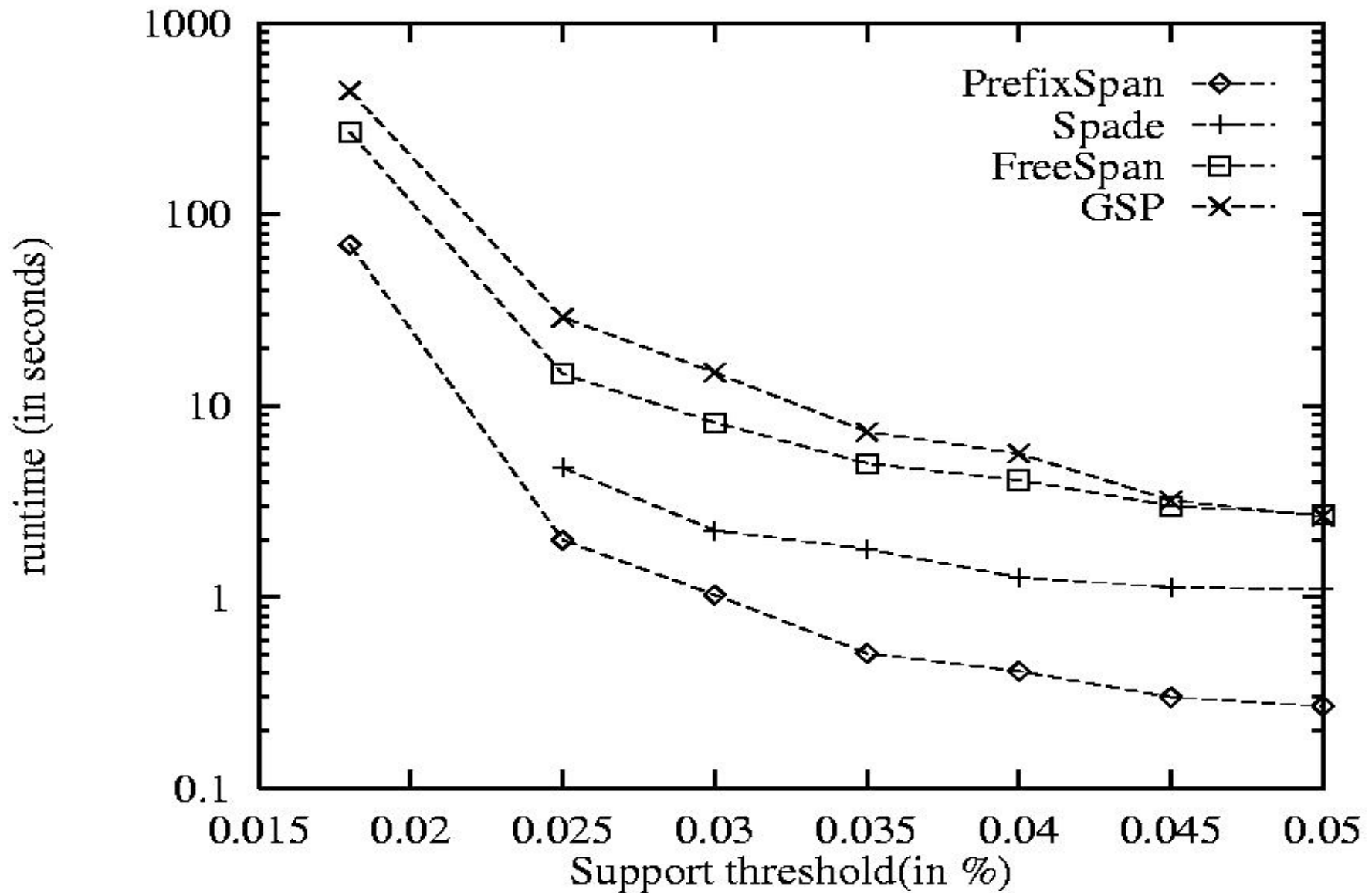
- Pseudo-projection avoids physically copying postfixes
  - Efficient in running time and space when database can be held in main memory
- However, it is not efficient when database cannot fit in main **memory**
  - Disk-based random accessing is very costly
- Suggested Approach:
  - Integration of physical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



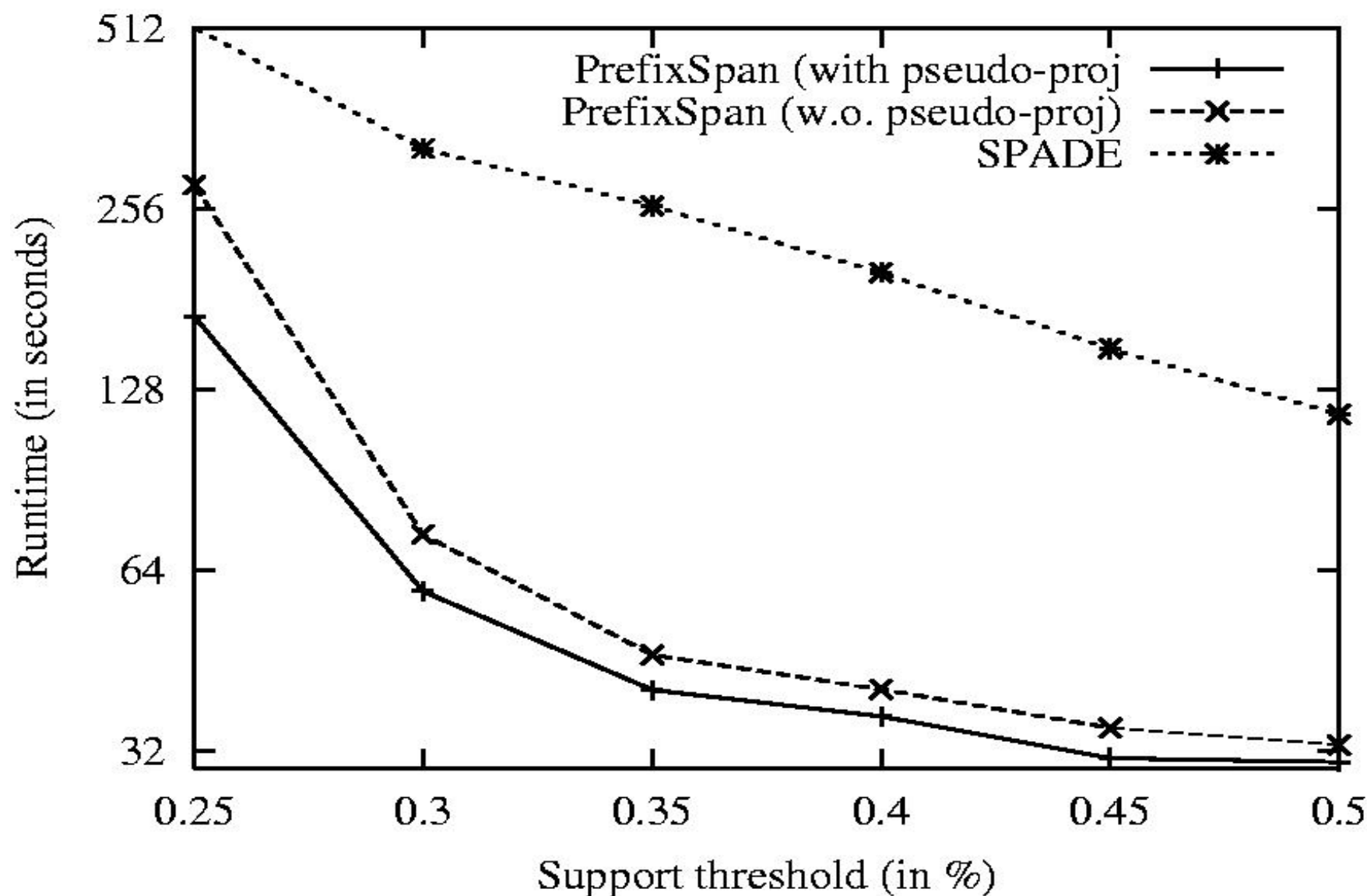
# Performance on Data Set C10T8S8I8



# Performance on Data Set Gazelle

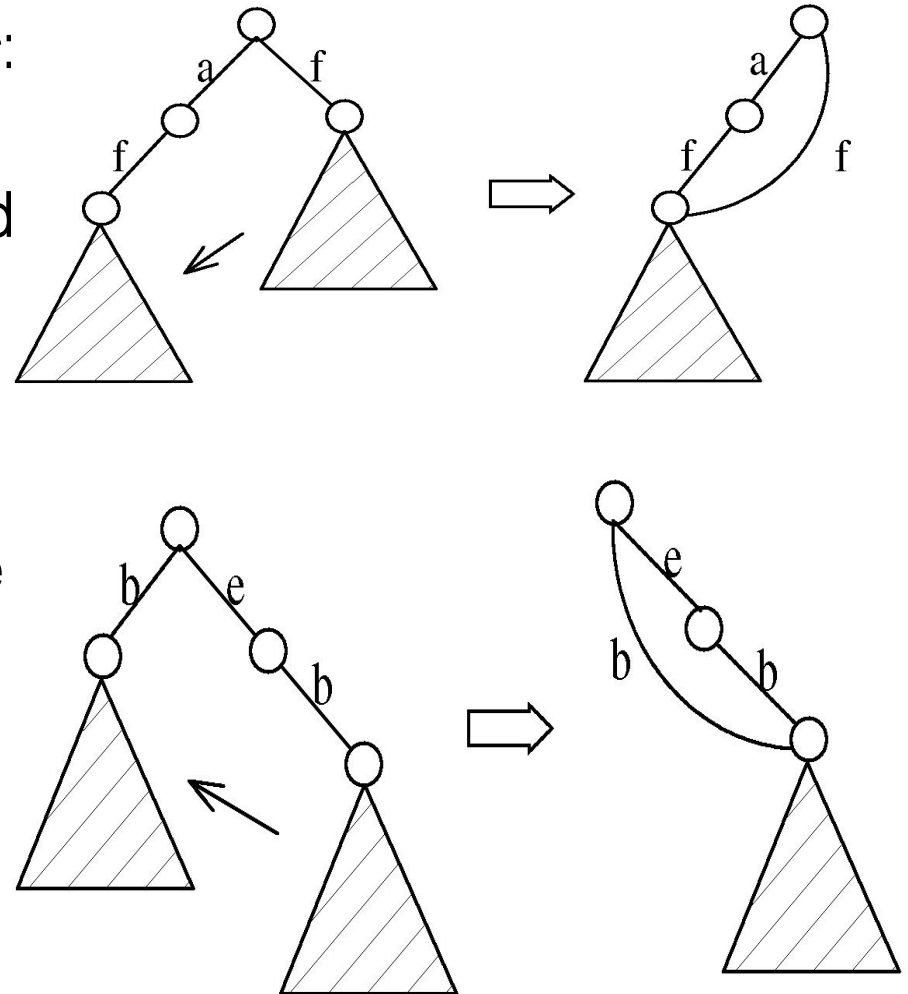


# Effect of Pseudo-Projection

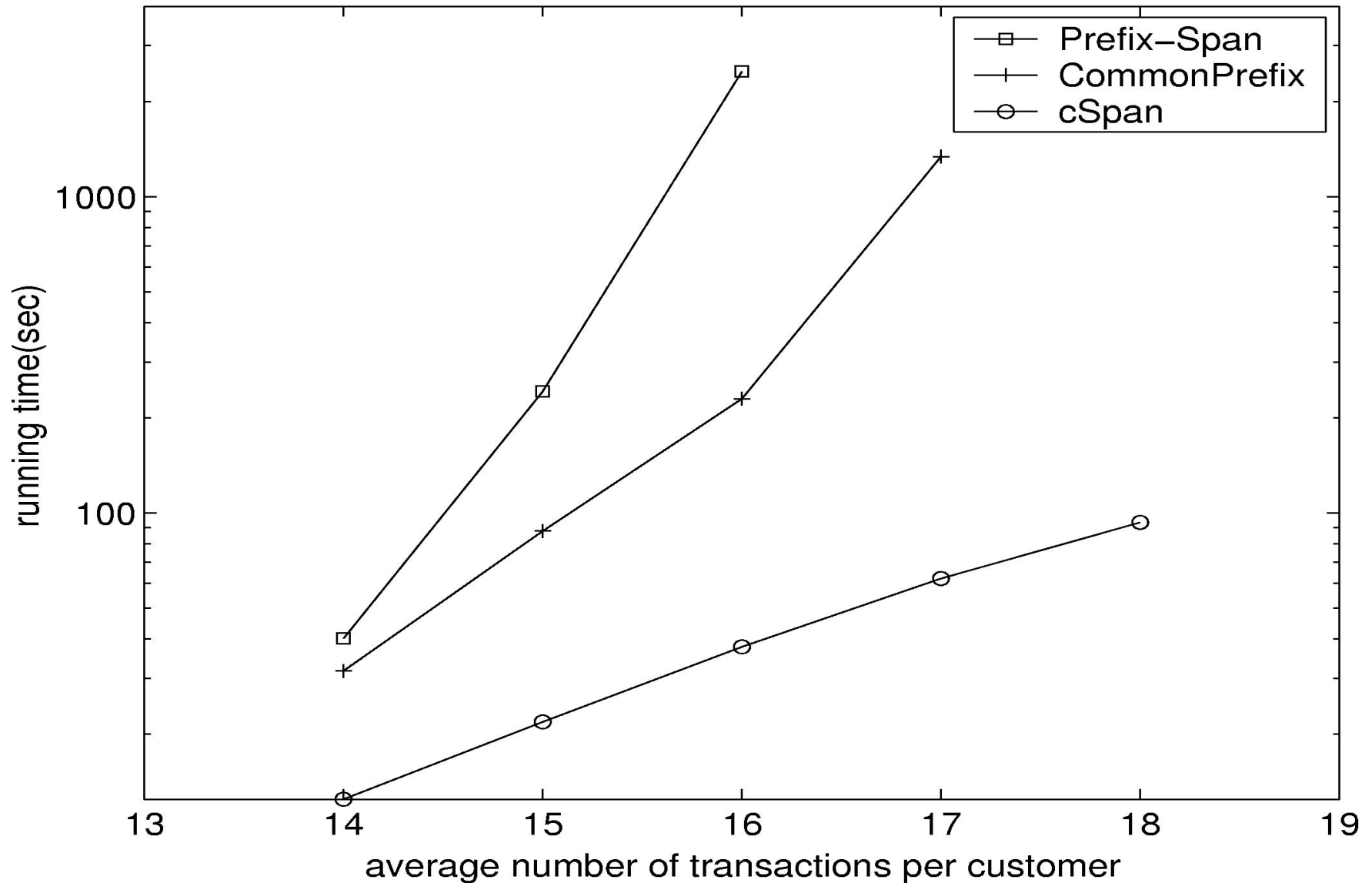


# CloSpan: Mining Closed Sequential Patterns

- A **closed sequential pattern**  $s$ : there exists no superpattern  $s'$  such that  $s' \supset s$ , and  $s'$  and  $s$  have the same support
- Motivation: reduces the number of (redundant) patterns but attains the same expressive power
- Using **Backward Subpattern and Backward Superpattern pruning** to prune redundant search space



# CloSpan: Performance Comparison with PrefixSpan



# Constraints for Seq.-Pattern Mining

---

- Item constraint
  - Find web log patterns only about online-bookstores
- Length constraint
  - Find patterns having at least 20 items
- Super pattern constraint →
  - Find super patterns of “PC digital camera”
- Aggregate constraint
  - Find patterns that the average price of items is over \$100

# More Constraints

---

- Regular expression constraint
  - Find patterns “starting from Yahoo homepage, search for hotels in Washington DC area”
  - `Yahootravel(WashingtonDC|DC)(hotel|motel|lodging)`
- Duration constraint
  - Find patterns about  $\pm 24$  hours of a shooting
- Gap constraint
  - Find purchasing patterns such that “the gap between each consecutive purchases is less than 1 month”

# From Sequential Patterns to Structured Patterns

---

- Sets, sequences, trees, graphs, and other structures
  - Transaction DB: Sets of items
    - $\{\{i_1, i_2, \dots, i_m\}, \dots\}$
  - Seq. DB: Sequences of sets:
    - $\{\langle\{i_1, i_2\}, \dots, \{i_m, i_n, i_k\}\rangle, \dots\}$
  - Sets of Sequences:
    - $\{\{\langle i_1, i_2 \rangle, \dots, \langle i_m, i_n, i_k \rangle\}, \dots\}$
  - Sets of trees:  $\{t_1, t_2, \dots, t_n\}$
  - Sets of graphs (mining for frequent subgraphs):
    - $\{g_1, g_2, \dots, g_n\}$
- Mining structured patterns in XML documents, biochemical structures, etc.



# Episodes and Episode Pattern Mining

---

- Other methods for specifying the kinds of patterns
  - Serial episodes:  $A \rightarrow B$
  - Parallel episodes:  $A \ \& \ B$
  - Regular expressions:  $(A \mid B)C^*(D \rightarrow E)$
- Methods for episode pattern mining
  - Variations of Apriori-like algorithms, e.g., **GSP**
  - Database projection-based pattern growth
    - Similar to the frequent pattern growth without candidate generation

# Periodicity Analysis

---

- Periodicity is everywhere: tides, seasons, daily power consumption, etc.
- Full periodicity
  - Every point in time contributes (precisely or approximately) to the periodicity
- Partial periodicity: A more general notion
  - Only some segments contribute to the periodicity
    - Jim reads NY Times 7:00-7:30 am every week day
- Cyclic association rules
  - Associations which form cycles
- Methods
  - Full periodicity: FFT, other statistical analysis methods
  - Partial and cyclic periodicity: Variations of Apriori-like mining methods

# Summary

---

- Sequential Pattern Mining is useful in many application, e.g. weblog analysis, financial market prediction, BioInformatics, etc.
- It is similar to the frequent itemsets mining, *but* with consideration of ordering.
- We have looked at different approaches that are descendants from two popular algorithms in mining frequent itemsets
  - Candidates Generation: AprioriAll and GSP
  - Pattern Growth: FreeSpan and PrefixSpan