Sequential Pattern Mining

What is Sequential Pattern Mining

- Find patterns in data where items are delivered in a **sequence**.
- A sequence is an ordered list of events

- Examples include:
  - Time series data
  - Symbolic sequences
  - Biological sequences

Applications

- Applications of sequential pattern mining
  - Customer shopping sequences
  - Medical treatments
  - Natural disasters (e.g., earthquakes),
  - Science & engineering processes
  - Stocks and markets, etc.
  - Telephone calling patterns, Weblog click streams
  - DNA sequences and gene structures
  - Sports data mining

The Traditional CS Example

- A sequence database consists of ordered elements or events
- Transactions are orderless (e.g. diapers and beer)
- Sequences include some order (e.g. diapers then beer)
- 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(abc)(ac)d(cf)>
20	<(ad)c(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

Subsequence vs. super sequence

- A sequence is an ordered list of events, denoted $< e_1\ e_2 \dots e_l >$

- Given two sequences $\alpha =< a_1\ a_2 \dots a_n >$ and $\beta =< b_1\ b_2 \dots b_m >$

- α is called a subsequence of β, denoted as $\alpha \subseteq \beta$, if there exist integers $1 \leq j_1 < j_2 < \dots < j_n \leq m$ such that $a_1 \subseteq b_{j\_1}$, $a_2 \subseteq b_{j\_2}, \dots, a_n \subseteq b_{j\_n}$

- β is a super sequence of α
  - E.g. α=< (ab), d> and β=< (abc), (de), (abc)>

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

An element may contain a set of items e.g. (ef). 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 in 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** and **scalable** involving only a small number of database scans
  - be able to incorporate various kinds of user-specific constraints

Thought Exercise

- How can the **a-priori** property be extended to sequential patterns?
- What would an a-priori-like algorithm for sequential pattern mining look like?
  - Candidate generation?

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 Leanining'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 & Sirkant'94)
  - If a sequence *S* is not frequent, then none of the super-sequences of *S* is frequent
  - E.g, <hb> is infrequent so are <hab> and <(ah)b>

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)>

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)>
<f>						

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

Apriori prunes 44.57% candidates

Finding Lenth-2 Sequential Patterns

- Scan updated 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

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 Mining Process

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

Cand. cannot pass sup. threshold

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

Cand. not in DB at all

3rd scan: 46 cand. 19 length-3 seq. 20 cand. not in DB at all <abb> <aab> <aba> <baa> <bab> …

2nd scan: 51 cand. 19 length-2 seq. 10 cand. not in DB at all <aa> <ab> … <af> <ba> <bb> … <ff> <(ab)> … <(ef)>

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

F_1 = the set of frequent 1-sequence

$k=2$,

while $F(k-1)$ is not empty;

Generate candidate sets C_k (set of candidate k-sequences);

For all input sequences *s* in the database *D*

Increment count of all a in C_k if s supports a

$F_k = \{a \in C_k$ such that its frequency exceeds the threshold$\}$

$k = k+1$;

Result = Set of all frequent sequences is the union of all F_ks

End

End

The GSP Algorithm

- Benefits from Apriori pruning
  - Reduces search space

- Bottlenecks
  - Scans the database multiple times
  - Generates a huge set of candidate sequences

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)

- Given sequence <a(abc)(ac)d(cf)>
- <a>, <aa>, <a(ab)> and <a(abc)> are *prefixes* of sequence <a(abc)(ac)d(cf)>

Prefix	*Suffix* (Prefix-Based *Projection)*
<a>	<(abc)(ac)d(cf)>
<aa>	<(_bc)(ac)d(cf)>
<ab>	<(_c)(ac)d(cf)>

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 seq. pat. 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>

Finding Sequence Patterns with Prefix <a>

- Only need to consider projections with respect to <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>, <(ac)>, <ad>, <ae>, <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>

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

Length-2 sequential patterns
<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>

The Algorithm of PrefixSpan

- **Input**: A sequence database S, and the minimum support threshold min_sup
- **Output**: The complete set of sequential patterns
- **Method**: Call PrefixSpan(<>,0,S)
- **Subroutine** PrefixSpan(α, l, S|α)
- **Parameters**:
  - α: sequential pattern,
  - l: the length of α;
  - S|α: the α-projected database, if α ≠<>; otherwise; the sequence database S

The Algorithm of PrefixSpan

- **Method**

1. Scan S|α once, find the set of frequent items b such that:

    a) b can be assembled to the last element of α to form a sequential pattern;

    b) <b> can be appended to α to form a sequential pattern.

2. For each frequent item b, append it to α to form a sequential pattern α’, and output α’;

3. For each α’, construct α’-projected database S|α’, and call PrefixSpan(α’, l+1, S|α’).

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 (disk) vs. pseudo-projection (memory)
  - 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=<a(abc)(ac)d(cf)>

↓ <a>

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

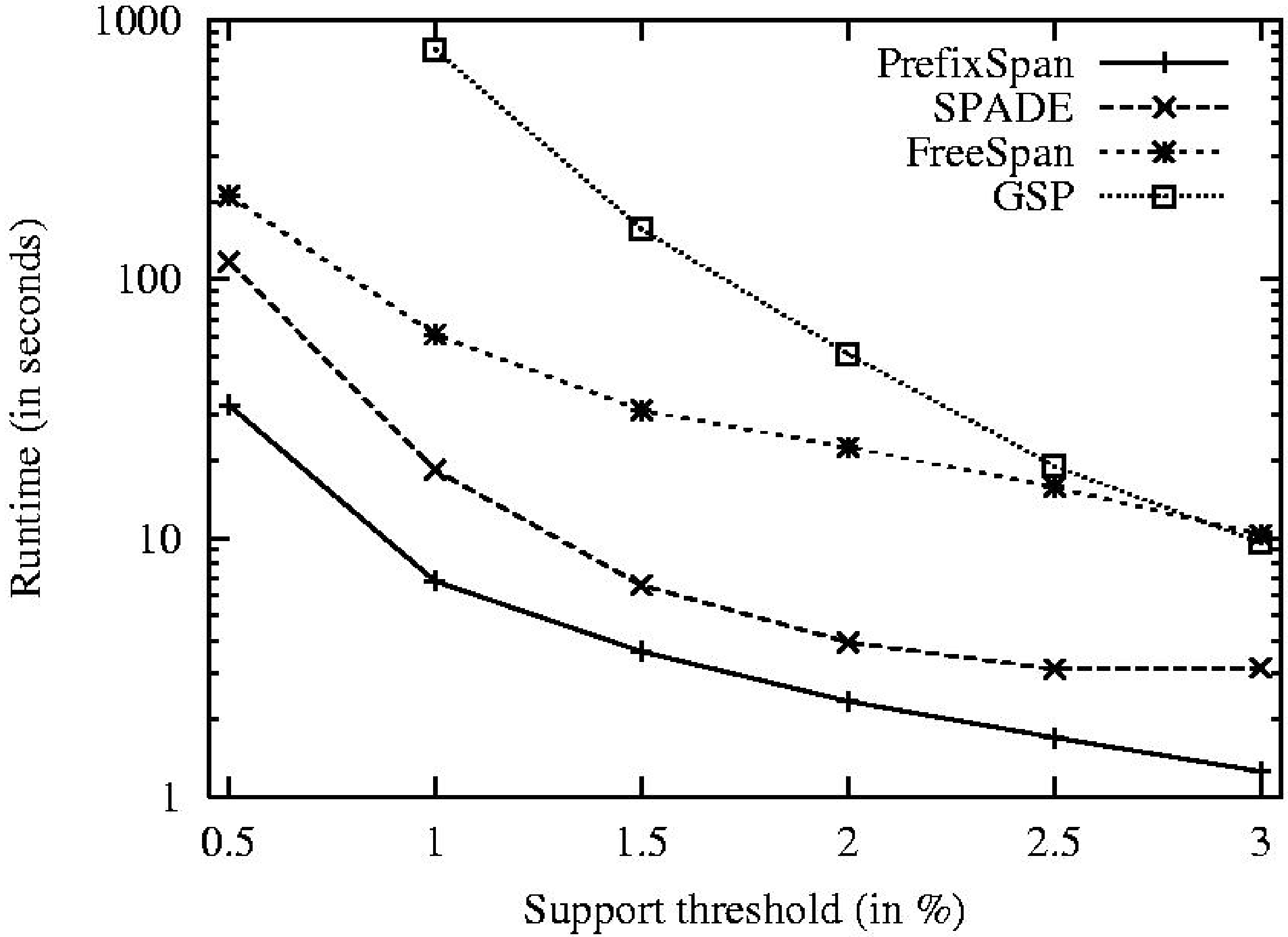
↓ <ab>

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

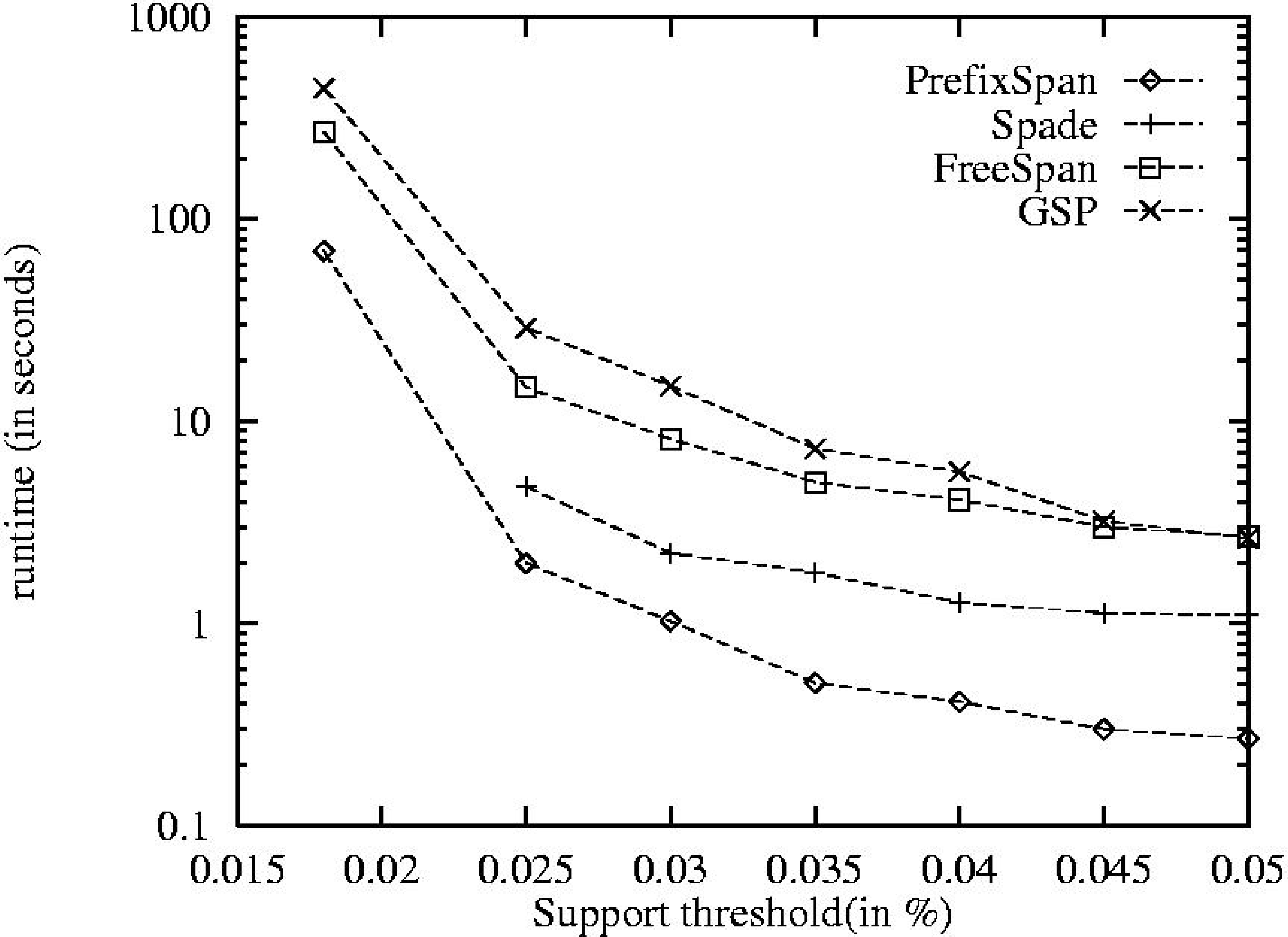
Pseudo-Projection vs. Physical Projection

- Pseudo-projection avoids physically copying 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

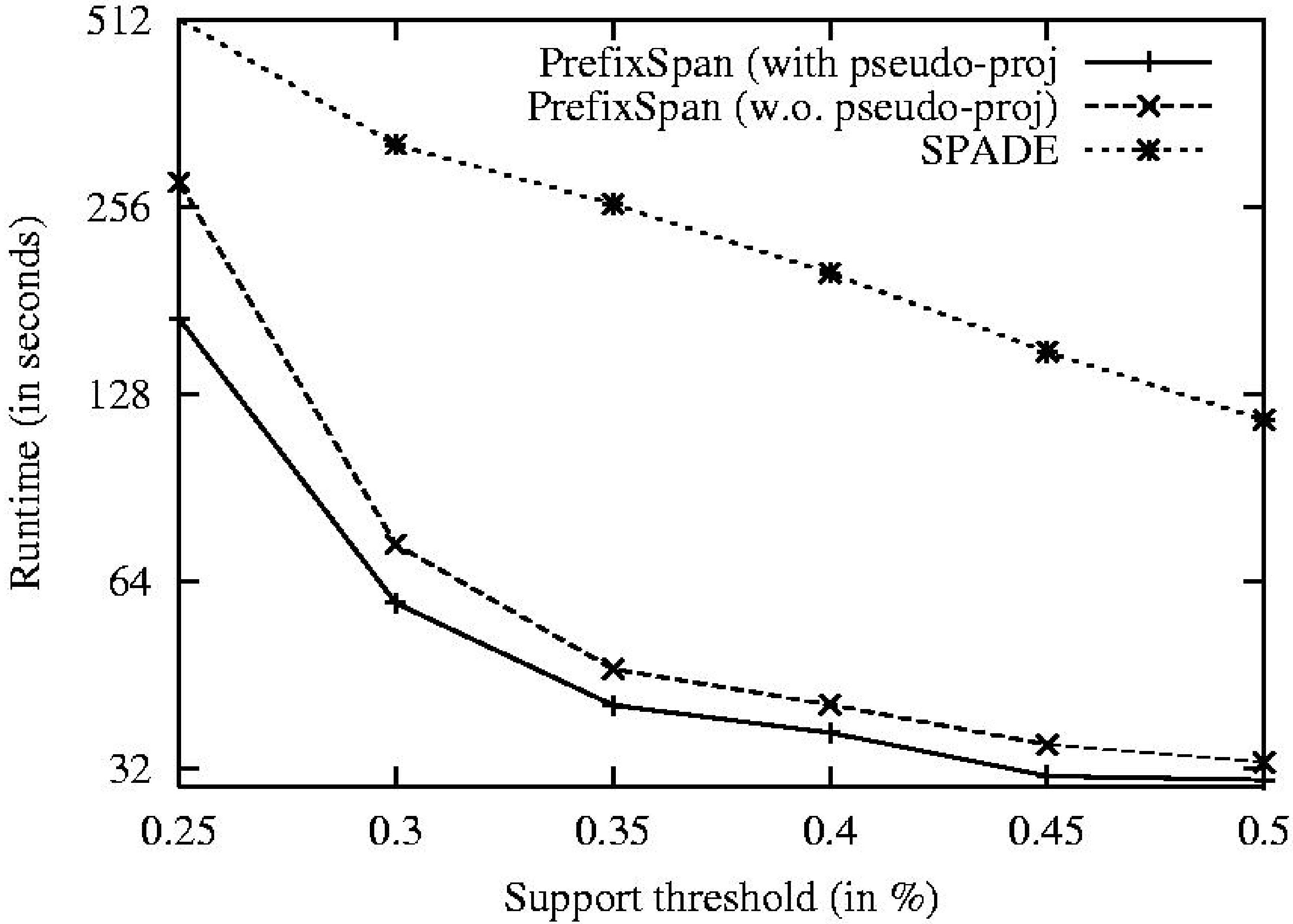
Performance on Data Set C10T8S8I8



Performance on Data Set Gazelle

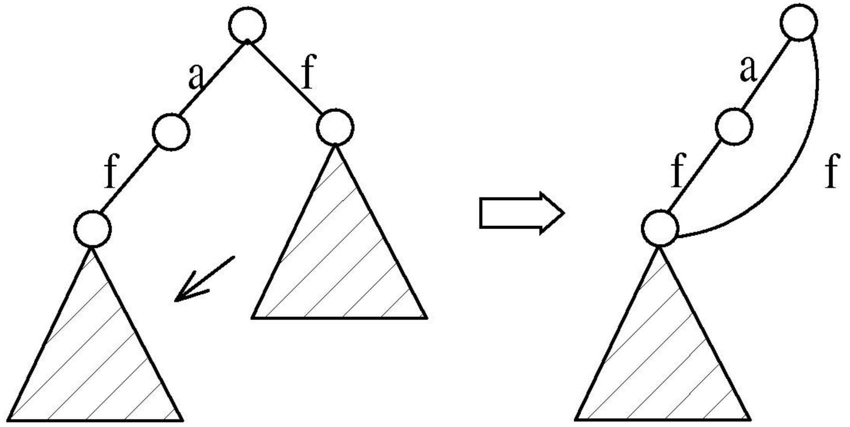


Effect of Pseudo-Projection

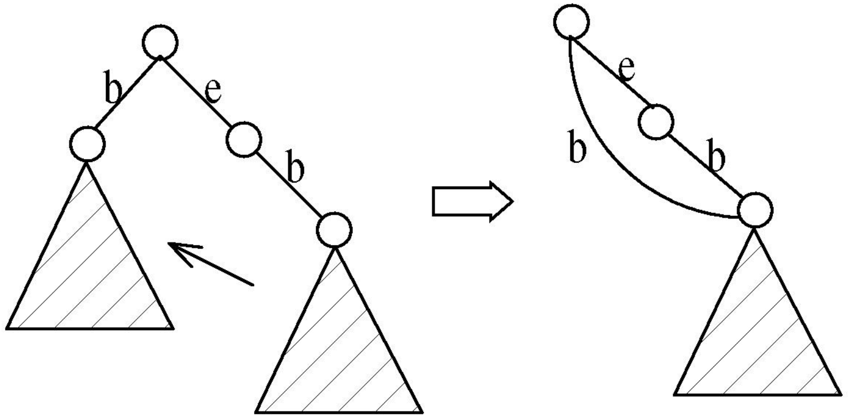


CloSpan: Mining Closed Sequential Patterns

- 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

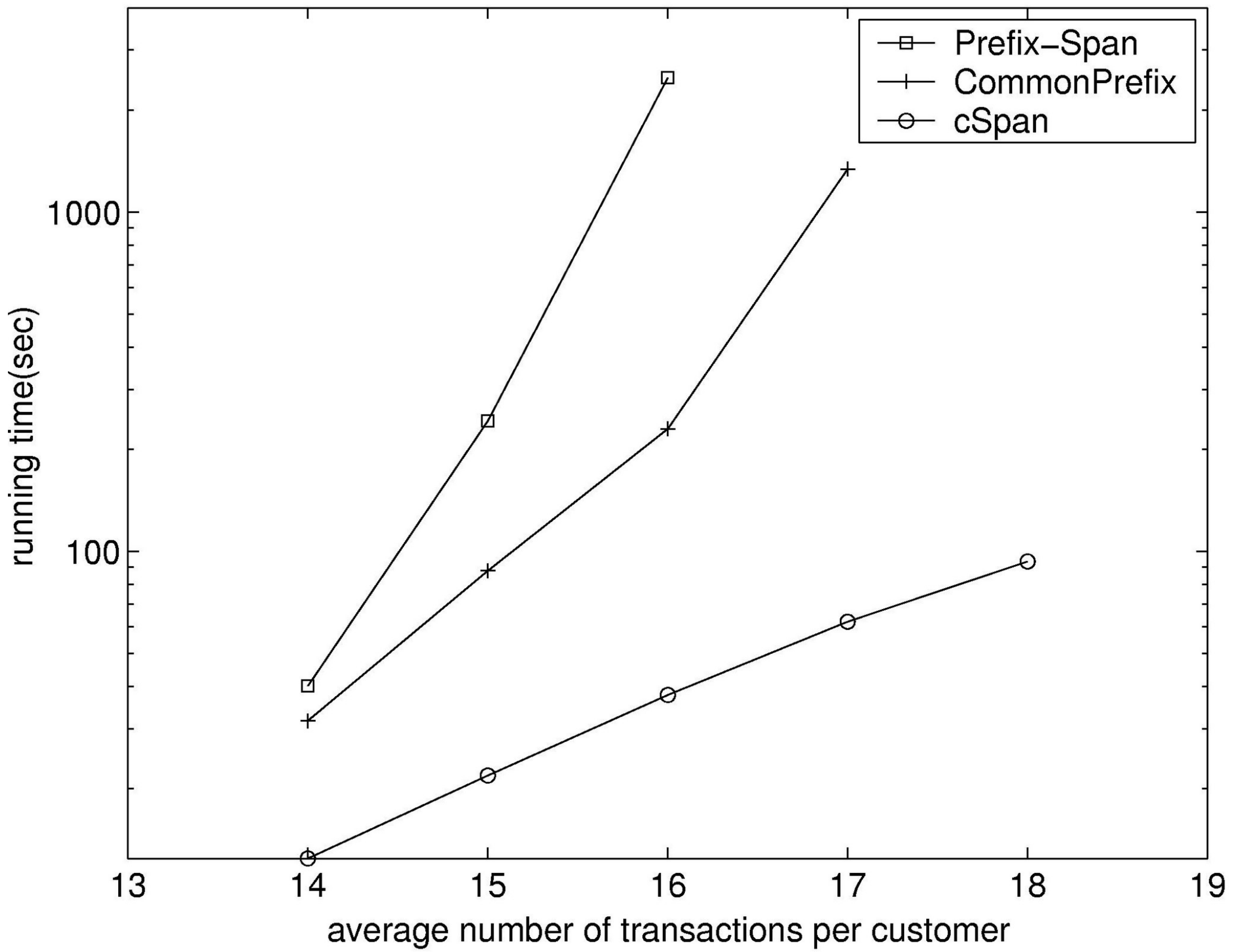


Backward Sub-pattern



Backward Super-pattern

CloSpan: Performance Comparison with PrefixSpan



Extensions to Frequent Sequence Mining

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 ±24 hours of an event

- 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
    - $\{\{i_1, i_2, \ldots, i_m\}, \ldots\}$
  - Seq. DB: Sequences of sets:
    - $\{<\{i_1, i_2\}, \ldots, \{i_m, i_n, i_k\}>, \ldots\}$
  - Sets of Sequences:
    - $\{\{<i_1, i_2>, \ldots, <i_m, i_n, i_k>\}, \ldots\}$
  - Sets of trees: $\{t_1, t_2, \ldots, t_n\}$
  - Sets of graphs (mining for frequent subgraphs):
    - $\{g_1, g_2, \ldots, g_n\}$
- Mining structured patterns in XML documents, bio-chemical 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 | 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