CS145: INTRODUCTION TO DATA MINING

Sequence Data: Sequential Pattern Mining

Instructor: Yizhou Sun

yzsun@cs.ucla.edu

November 24, 2020

Methods to Learn

	Vector Data	Set Data	Sequence Data	Text Data
Classification	**Logistic Regression; Decision Tree**; **KNN; SVM**; **NN**			Naïve Bayes for Text
Clustering	**K-means; hierarchical clustering; DBSCAN; Mixture Models**			PLSA
Prediction	**Linear Regression** GLM*			
Frequent Pattern Mining		**Apriori; FP growth**	**GSP; PrefixSpan**	
Similarity Search			DTW	

Sequence Data

- Introduction
- GSP
- PrefixSpan
- Summary

Sequence Database

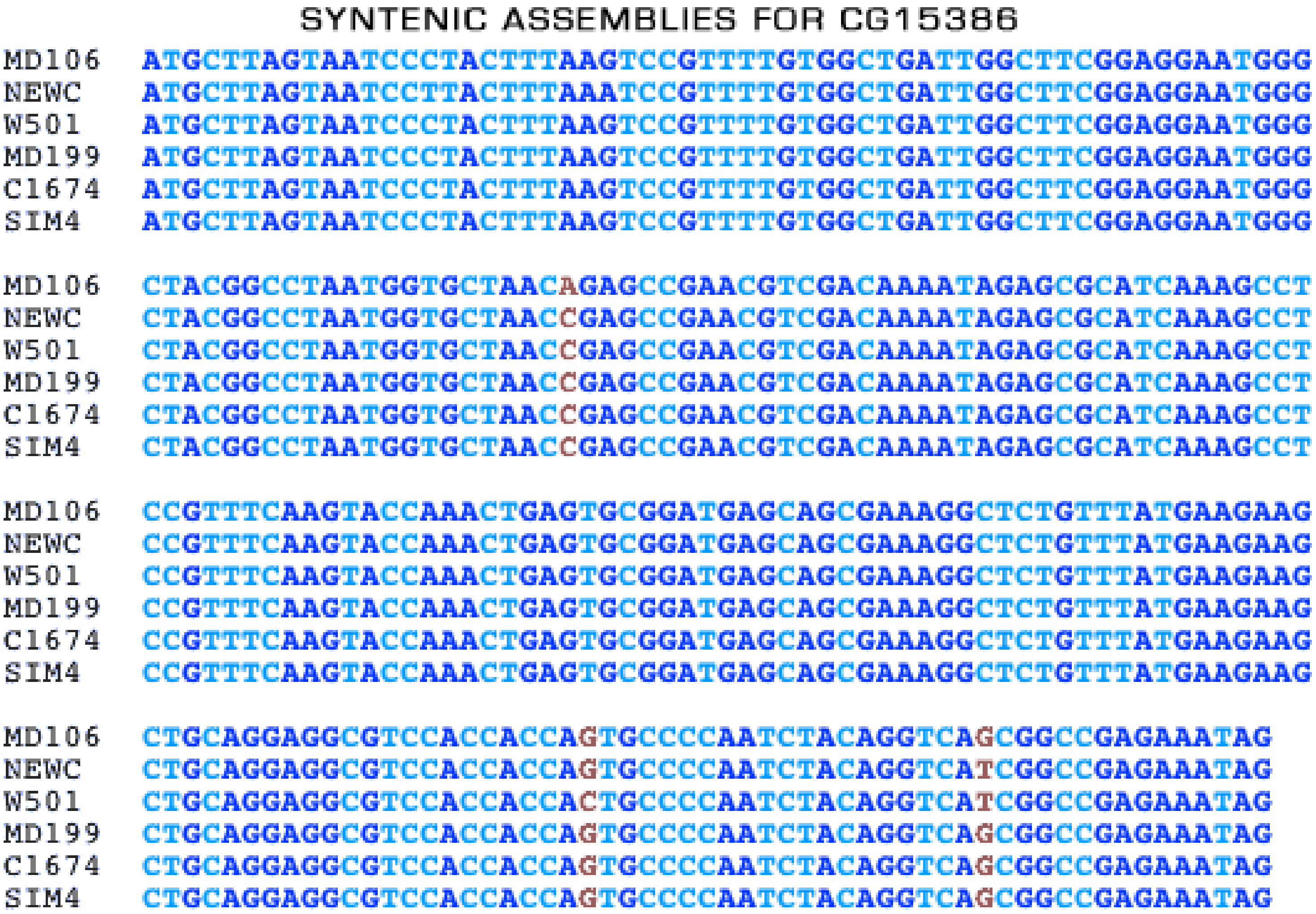
- A sequence database consists of sequences of ordered elements or events, recorded with or without a concrete notion of time.

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

Example: Music

- Music: midi files

Example: DNA Sequence



```
                SYNTENIC ASSEMBLIES FOR CG15386
MD106   ATGCTTAGTAATCCCTACTTTAAGTCCGTTTTGTGGCTGATTGGCTTCGGAGGAATGGG
NEWC    ATGCTTAGTAATCCTTACTTTAAATCCGTTTTGTGGCTGATTGGCTTCGGAGGAATGGG
W501    ATGCTTAGTAATCCCTACTTTAAGTCCGTTTTGTGGCTGATTGGCTTCGGAGGAATGGG
MD199   ATGCTTAGTAATCCCTACTTTAAGTCCGTTTTGTGGCTGATTGGCTTCGGAGGAATGGG
C1674   ATGCTTAGTAATCCCTACTTTAAGTCCGTTTTGTGGCTGATTGGCTTCGGAGGAATGGG
SIM4    ATGCTTAGTAATCCCTACTTTAAGTCCGTTTTGTGGCTGATTGGCTTCGGAGGAATGGG

MD106   CTACGGCCTAATGGTGCTAACAGAGCCGAACGTCGACAAAATAGAGCGCATCAAAGCCT
NEWC    CTACGGCCTAATGGTGCTAACCGAGCCGAACGTCGACAAAATAGAGCGCATCAAAGCCT
W501    CTACGGCCTAATGGTGCTAACCGAGCCGAACGTCGACAAAATAGAGCGCATCAAAGCCT
MD199   CTACGGCCTAATGGTGCTAACCGAGCCGAACGTCGACAAAATAGAGCGCATCAAAGCCT
C1674   CTACGGCCTAATGGTGCTAACCGAGCCGAACGTCGACAAAATAGAGCGCATCAAAGCCT
SIM4    CTACGGCCTAATGGTGCTAACCGAGCCGAACGTCGACAAAATAGAGCGCATCAAAGCCT

MD106   CCGTTTCAAGTACCAAACTGAGTGCGGATGAGCAGCGAAAGGCTCTGTTTATGAAGAAG
NEWC    CCGTTTCAAGTACCAAACTGAGTGCGGATGAGCAGCGAAAGGCTCTGTTTATGAAGAAG
W501    CCGTTTCAAGTACCAAACTGAGTGCGGATGAGCAGCGAAAGGCTCTGTTTATGAAGAAG
MD199   CCGTTTCAAGTACCAAACTGAGTGCGGATGAGCAGCGAAAGGCTCTGTTTATGAAGAAG
C1674   CCGTTTCAAGTACCAAACTGAGTGCGGATGAGCAGCGAAAGGCTCTGTTTATGAAGAAG
SIM4    CCGTTTCAAGTACCAAACTGAGTGCGGATGAGCAGCGAAAGGCTCTGTTTATGAAGAAG

MD106   CTGCAGGAGGCGTCCACCACCAGTGCCCCAATCTACAGGTCAGCGGCCGAGAAATAG
NEWC    CTGCAGGAGGCGTCCACCACCAGTGCCCCAATCTACAGGTCATCGGCCGAGAAATAG
W501    CTGCAGGAGGCGTCCACCACCACTGCCCCAATCTACAGGTCATCGGCCGAGAAATAG
MD199   CTGCAGGAGGCGTCCACCACCAGTGCCCCAATCTACAGGTCAGCGGCCGAGAAATAG
C1674   CTGCAGGAGGCGTCCACCACCAGTGCCCCAATCTACAGGTCAGCGGCCGAGAAATAG
SIM4    CTGCAGGAGGCGTCCACCACCAGTGCCCCAATCTACAGGTCAGCGGCCGAGAAATAG
```

Sequence Databases & Sequential Patterns

- Transaction databases vs. sequence databases
- Frequent patterns vs. (frequent) sequential patterns
- 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
  - Program execution sequence data sets
  - DNA sequences and gene structures

What Is Sequential Pattern Mining?

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

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

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

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>

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

Given *support threshold* *min_sup* =2, <(ab)c> is a *sequential pattern*

Sequence

- Event / element
  - An non-empty set of items, e.g., e=(ab)
- Sequence
  - An ordered list of events, e.g., $s = < e_1 e_2 \ldots e_l >$
- Length of a sequence
  - The number of instances of items in a sequence
  - The length of < (ef) (ab) (df) c b > is 8 (Not 5!)

Subsequence

- Subsequence
  - For two sequences $\alpha =< a_1 a_2 \ldots a_n >$ and $\beta =< b_1 b_2 \ldots b_m >$, α is called a subsequence of β if there exists integers $1 \leq j_1 < j_2 < \cdots < j_n \leq m$, such that $a_1 \subseteq b_{j_1}, \ldots, a_n \subseteq b_{j_n}$
- Supersequence
  - If α is a subsequence of β, β is a supersequence of α

e.g., <a(bc)dc> is a *subsequence* of <a(abc)(ac)d(cf)>

Sequential Pattern

- Support of a sequence α
  - Number of sequences in the database that are supersequence of α
  - $Support_S(\alpha)$
- α is frequent if $Support_S(\alpha) \geq \min\_support$
- A frequent sequence is called sequential pattern
  - l-pattern if the length of the sequence is l

Example

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>

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

Sequential Pattern Mining Algorithms

- 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)

Sequence Data

- Introduction
- GSP
- PrefixSpan
- Summary

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 do <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
  - proposed by Agrawal and Srikant, EDBT'96
- 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

- Examine GSP using an example
- Initial candidates: all singleton sequences
  - <a>, <b>, <c>, <d>, <e>, <f>, <g>, <h>
- Scan database once, count support for candidates

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

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)>

GSP: Generating 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>

51 length-2 Candidates

	<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

How to Generate Candidates in General?

- From L_{k-1} to C_k
- Step 1: join
  - s_1 *and* s_2 can join, if dropping first item in s_1 is the same as dropping the last item in s_2
  - Examples:
    - <(12)3> join <(2)34> = <(12)34>
    - <(12)3> join <(2)(34)> = <(12)(34)>
- Step 2: pruning
  - Check whether all length k-1 subsequences of a candidate is contained in L_{k-1}

The GSP Mining Process

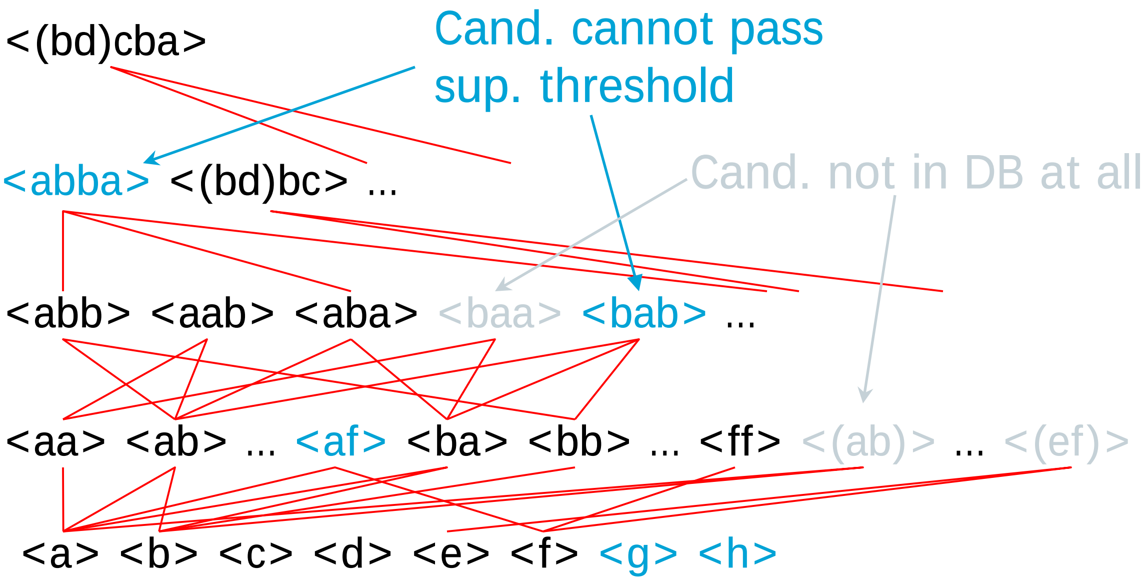
5th scan: 1 cand. 1 length-5 seq. pat.

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.



<(bd)cba>

Cand. cannot pass sup. threshold

<abba> <(bd)bc> ...

Cand. not in DB at all

<abb> <aab> <aba> <baa> <bab> ...

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

<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)>

Candidate Generate-and-test: Drawbacks

- A huge set of candidate sequences generated.
  - Especially 2-item candidate sequence.
- Multiple Scans of database needed.
  - The length of each candidate grows by one at each database scan.
- Inefficient for mining long sequential patterns.
  - A long pattern grow up from short patterns
  - The number of short patterns is exponential to the length of mined patterns.

*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
  - Item: <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			

Join two tables

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 GSP and SPADE

- A huge set of candidates could be generated
  - 1,000 frequent length-1 sequences generate s huge number of length-2 candidates!

$$1000\times1000+\frac{1000\times999}{2}=1{,}499{,}500$$

- Multiple scans of database in mining
- Breadth-first search
- Mining long sequential patterns
  - Needs an exponential number of short candidates
  - A length-100 sequential pattern needs 10^{30} candidate sequences!

$$\sum_{i=1}^{100}\binom{100}{i}=2^{100}-1\approx10^{30}$$

Sequence Data

- Introduction
- GSP
- PrefixSpan
- Summary

Prefix and Suffix

Assume a pre-specified order on items, e.g., alphabetical order

- <a>, <aa>, <a(ab)> and <a(abc)> are *prefixes* of sequence <a(abc)(ac)d(cf)>
  - Note <a(ac)> is not a prefix of <a(abc)(ac)d(cf)>
- Given sequence <a(abc)(ac)d(cf)>

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

  - (_bc) means: the last element in the prefix together with (bc) form one element

Prefix-based Projection

- Given a sequence, α, let α' be subsequence of α
  - α' is called a projection of α w.r.t. prefix β, if and only if
    - α' has prefix β, and
    - α' is the maximum subsequence of α with prefix β
  - Example:
    - <ad(cf)> is a projection of <a(abc)(ac)d(cf)> w.r.t. prefix <ad>

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

Projected (Suffix) Database

- Let α be a sequential pattern, α-projected database is the collection of suffixes of projections of sequences in the database w.r.t. prefix α
  - Examples
    - <a>-projected database
      - <(abc)(ac)d(cf)>
      - <(_d)c(bc)(ae)>
      - <(_b)(df)cb>
      - <(_f)cbc>
    - <ab>-projected database
      - <(_c)(ac)d(cf)> (<a(bc)(ac)d(cf)> is the projection of <a(abc)(ac)d(cf)> w.r.t. prefix <ab>)
      - <(_c)(ae)> (<a(bc)(ae)> is the projection of <(ad)c(bc)(ae)> w.r.t. prefix <ab>)
      - <c> (<abc> is the projection of <eg(af)cbc> w.r.t prefix <ab>)

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

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>
- Step 3: mine each subset recursively via

corresponding projected databases

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

Finding Seq. Patterns with Prefix <a>

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

- Only need to consider projections w.r.t. <a>
  - <a>-projected (suffix) 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>

Why are those 6 subsets?

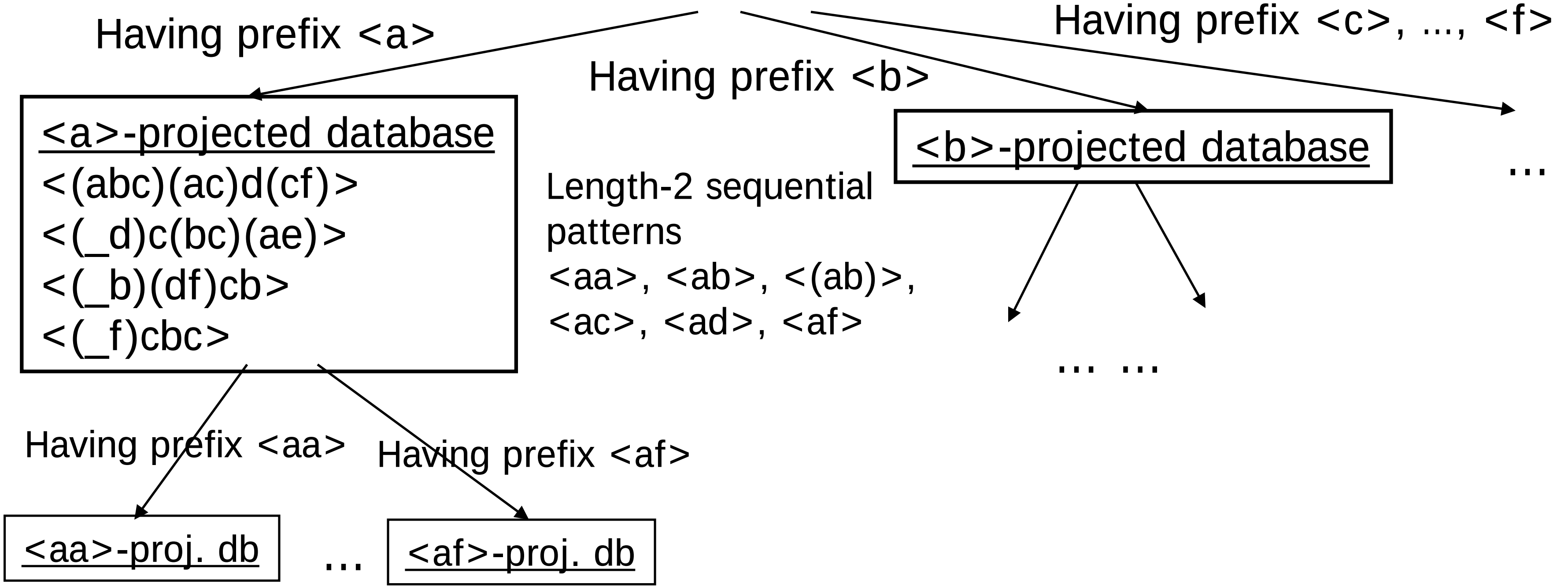
- By scanning the <*a*>-projected database once, its locally frequent items are identified as
  - $a : 2$, $b : 4$, $\_b : 2$, $c : 4$, $d : 2$, and $f : 2$.
- Thus all the length-2 sequential patterns prefixed with <*a*> are found, and they are:
  - $<aa> : 2$, $<ab> : 4$, $<(ab)> : 2$, $<ac> : 4$, $<ad> : 2$, and $<af> : 2$.

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>

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

<b>-projected database

…

… …

Having prefix <aa>

Having prefix <af>

<aa>-proj. db

…

<af>-proj. db

Examples

- <aa>-projected database
  - <(_bc)(ac)d(cf)>
  - <(_e)>
- <ab>-projected database
  - <(_c)(ac)d(cf)>
  - <(_c)(ae)>
  - <c>
- <(ab)>-projected database
  - <(_c)(ac)d(cf)>
  - <(df)cb>

<a>-projected database:

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

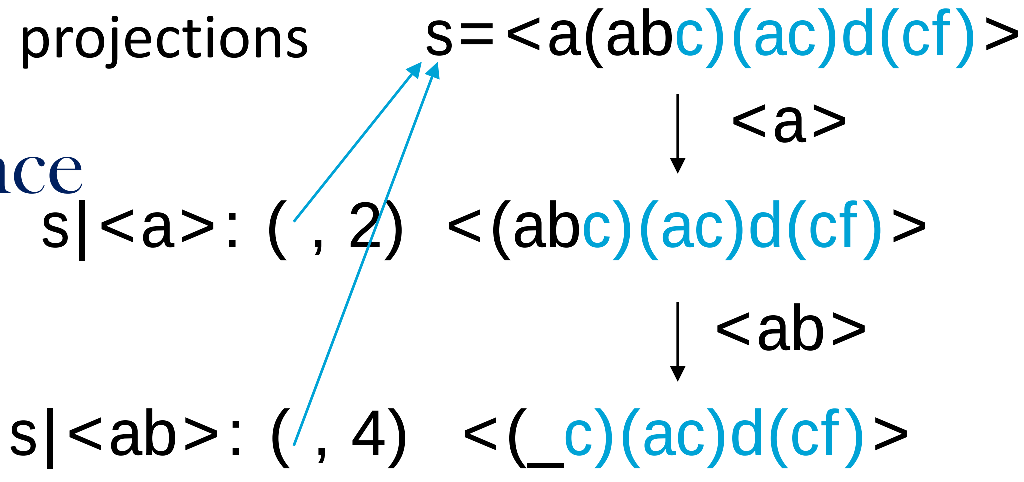
Reference: http://hanj.cs.illinois.edu/pdf/tkde04_spgjn.pdf

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
  - 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)>

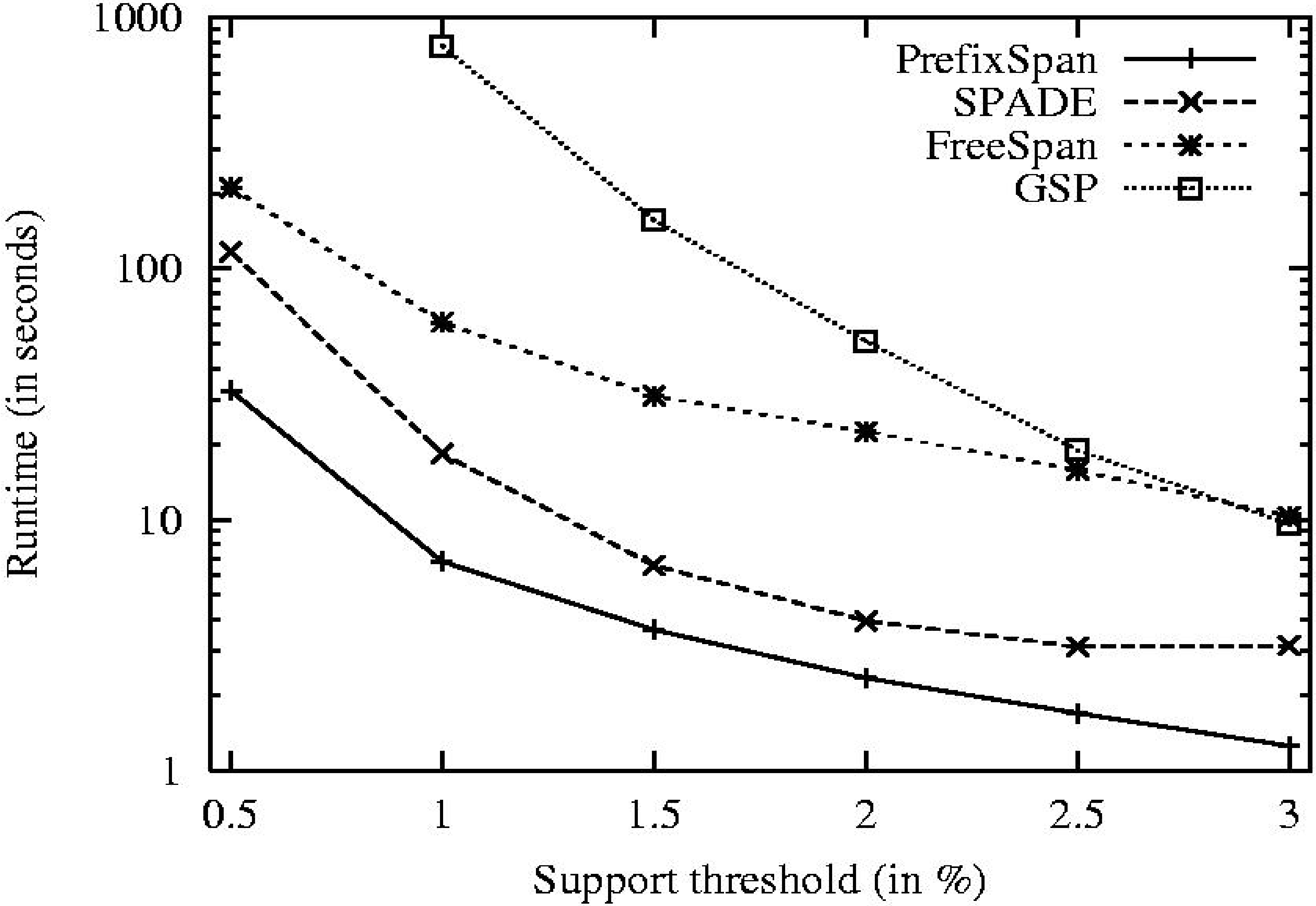
<ab>

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

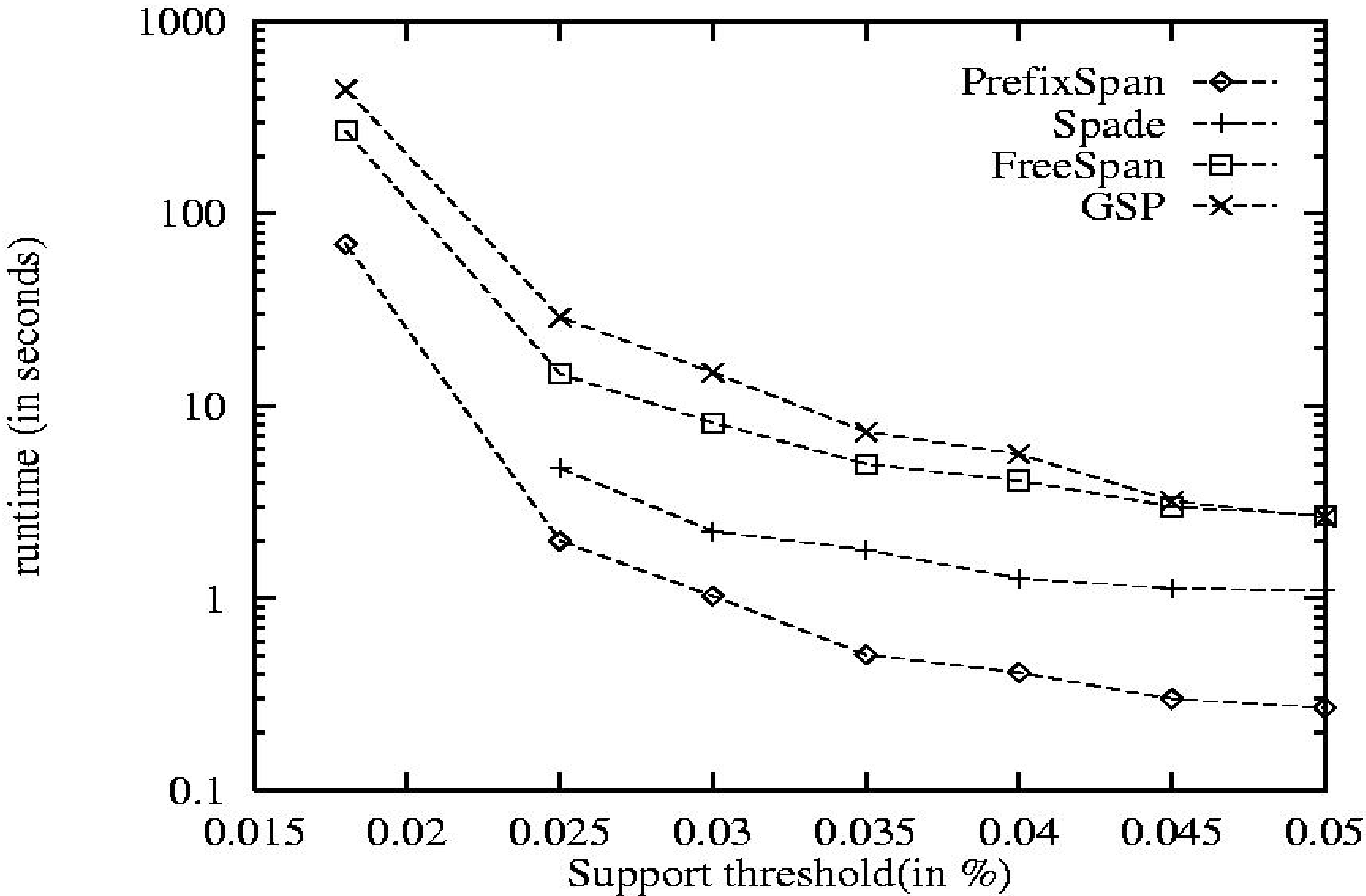
*Pseudo-Projection vs. Physical Projection

- 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

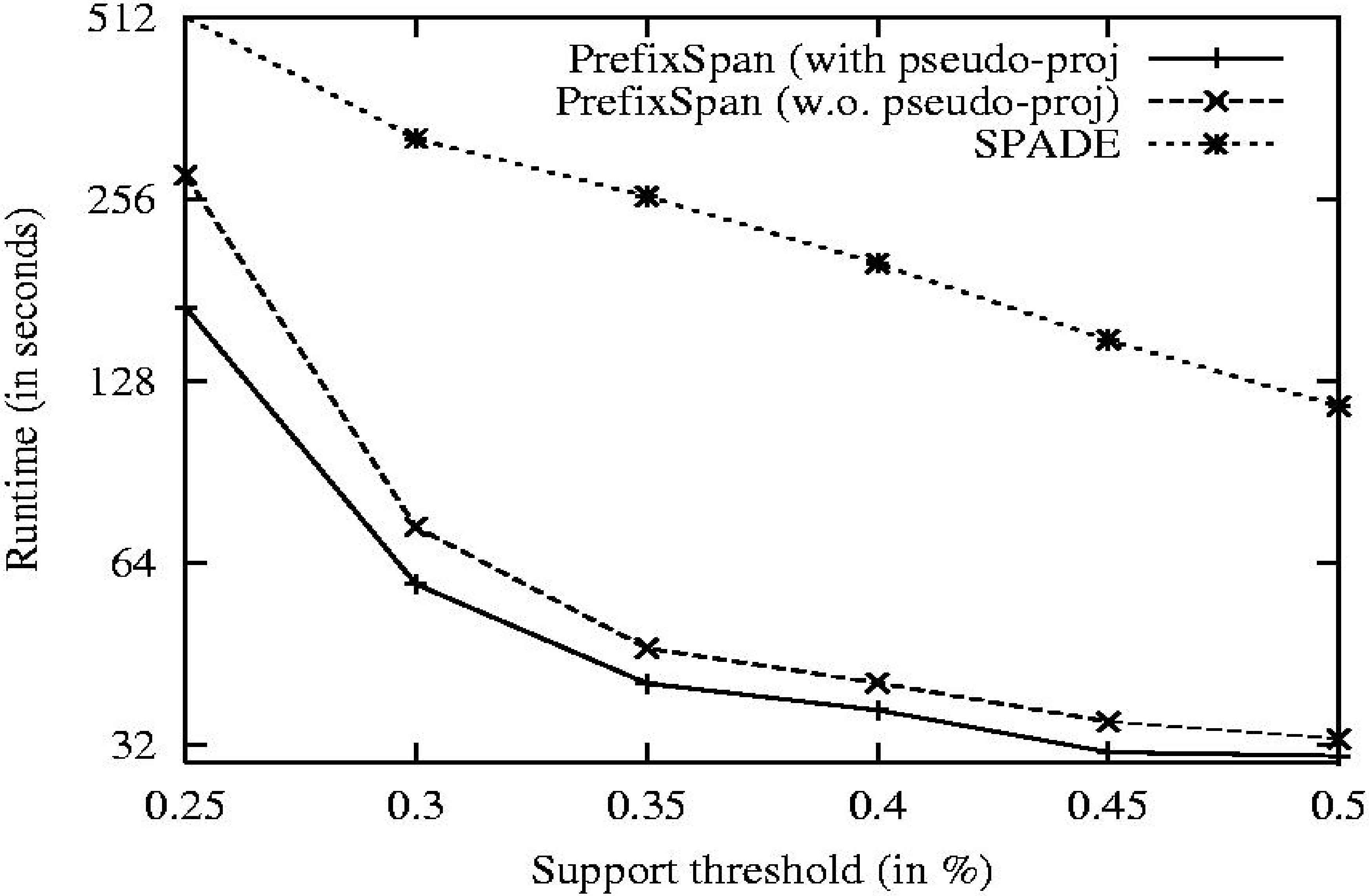
Performance on Data Set C10T8S8I8



Performance on Data Set Gazelle



*Effect of Pseudo-Projection



Sequence Data

- Introduction
- GSP
- PrefixSpan
- Summary

Summary

- Sequential Pattern Mining
    - GSP, PrefixSpan