Data Mining: Principle and Algorithms

-Chapter 3.6-
-Sequential Pattern Mining-

Jianyong Wang

Department of Computer Science and Technology
Tsinghua University, Beijing, China
Email: jianyong@tsinghua.edu.cn

Chapter 3 Mining Frequent Patterns, Association and Correlations

§ 3.1 Basic concepts and a road map

§ 3.2 Efficient and scalable frequent itemset mining methods

§ 3.3 Mining various kinds of association rules

§ 3.4 From association mining to correlation analysis

§ 3.5 Constraint-based association mining

§ 3.6 Sequential pattern mining

- *Frequent sequence mining*
- *Closed sequence mining*
- *Typical applications of frequent sequence mining*

§ 3.7 Graph pattern mining

§ 3.8 Summary

Outline

- Problem statement and motivation
- An overview of the current solutions
  - *GSP, SPADE, PrefixSPan, CloSpan*
- The BIDE algorithm
  - *Bi-Directional Extension* closure checking scheme
  - *BackScan* search space pruning
  - *ScanSkip* optimization
- Some typical applications

Problem Statement

- Frequent subsequence mining from a sequence DB
  - S is frequent if its support is no smaller than a minimum support
- Closed subsequence mining
  - S is closed if none of its super-sequence has the same support as S

A *sequence database*

SID	Sequence
10	< C A A B C >
20	< A B C B >
30	< C A B C >
40	< A B B C A >

<C C> is a *subsequence* of <C A A B C>, and <C A B C>

Given a support threshold *min_sup* =2, <C C> is a *frequent sequence*, but it is not *closed*, while <C A B C> is a frequent closed sequence.

Motivation: why mining frequent sequences?

- Different kinds of sequence databases
  - Customer shopping sequences, Web click-streams, DNA sequences, production and engineering processes, earthquake, and biological evolutions; and so on
- Various applications
  - Association/causality analysis
  - Frequent-sequence based classification
  - Frequent-sequence based clustering
  - Sequence based hot XML query pattern mining
  - … …

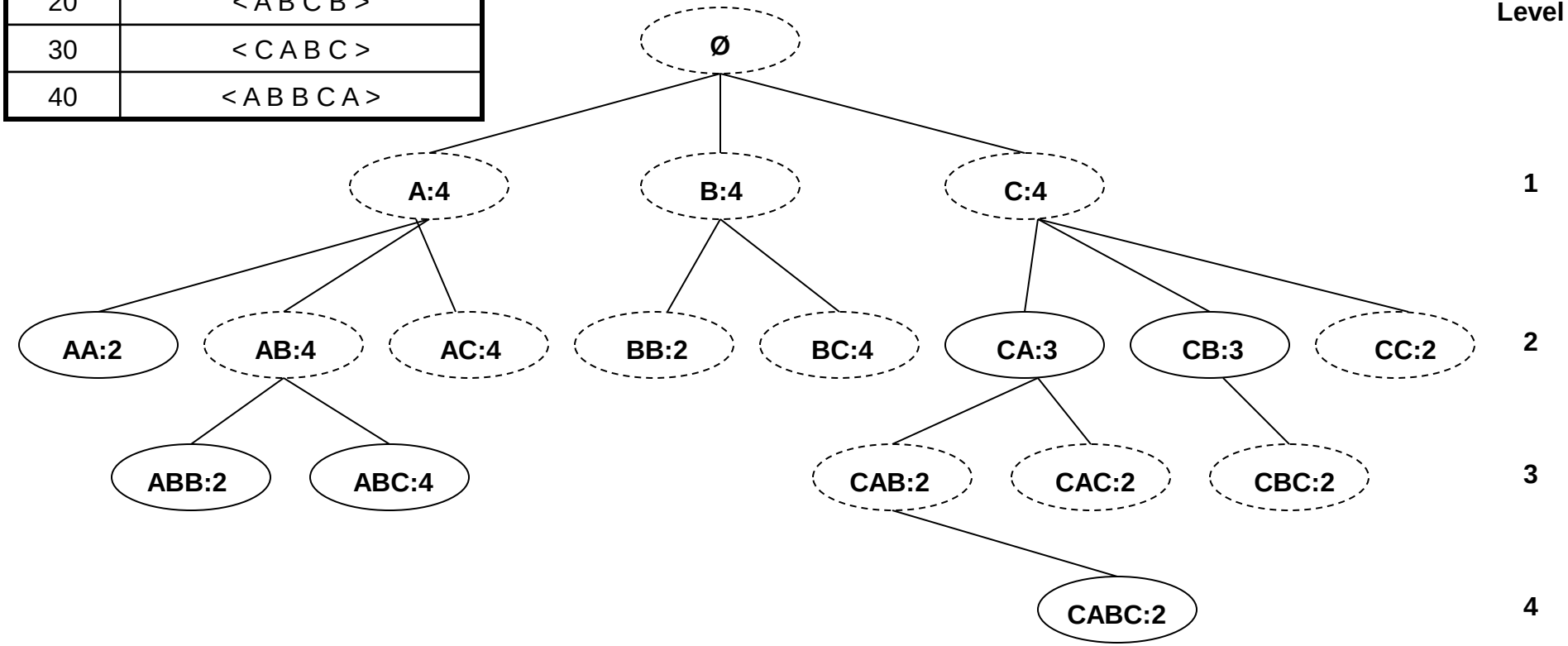
Motivation: why mining closed sequences?

- All the subsequences of a long frequent sequence must be frequent—Apriori property
  - If $(a_1, \ldots, a_{64})$ is frequent, it will generate $2^{64}-1$ frequent subsequences—What an exponential growth!
- More concise result set
  - The set of closed frequent sequences can be orders of magnitude smaller than the set of frequent sequences
- More efficient algorithm
  - Pruning unpromising search space

Search Space of Sequence Mining

- Assume item ordering: A ≤ B ≤ C, the complete search space of sequence mining forms a sequence tree

SID	Sequence
10	< C A A B C >
20	< A B C B >
30	< C A B C >
40	< A B B C A >



Comments on the Current Solutions

- Sequential pattern mining, e.g., GSP, SPADE, PrefixSpan, and SPAM
  - Generates huge number of patterns when min-sup is low
- Closed sequential pattern mining, e.g., CloSpan, BIDE
  - Generates much more concise result set
  - CloSpan vs. BIDE
    - *CloSpan* needs to maintain the set of closed pattern candidates and use a post-processing to remove the non-closed patterns
    - *Clospan* has high space and time complexity with low support

GSP—A Generalized Sequential Pattern Mining Algorithm

- 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 based on the Apriori property
  - 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

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

The GSP Mining Process

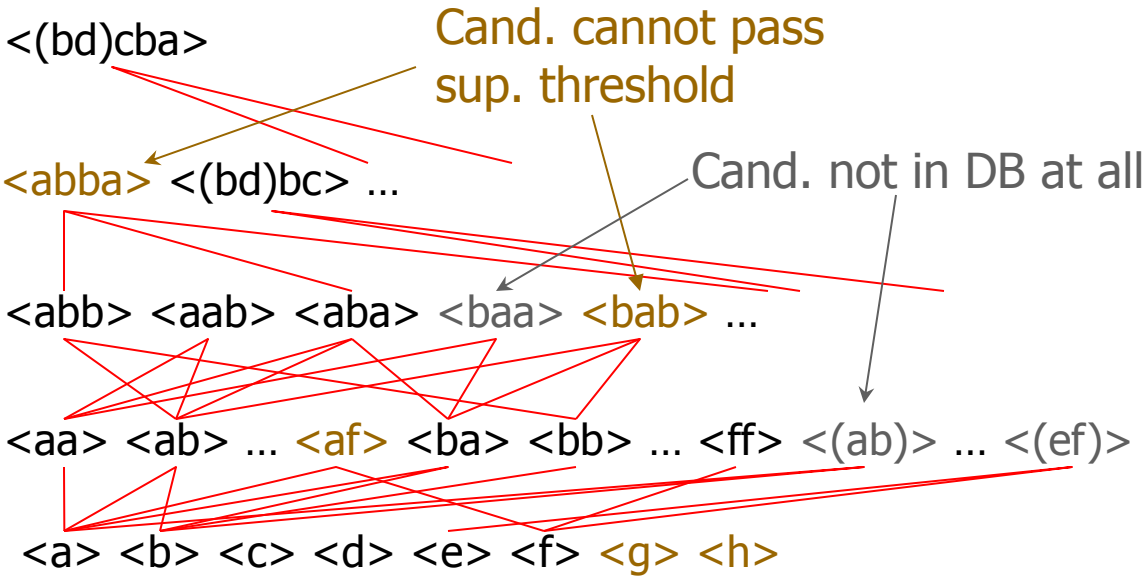
5th scan: 1 cand. 1 length-5 seq. pat.

4th scan: 8 cand. 6 length-4 seq. pat.

3rd scan: 46 cand. 19 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.



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 grows 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, Machine Learning.
- 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			

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

$$1000 \times 1000 + \frac{1000 \times 999}{2} = 1{,}499{,}500$$ length-2 candidates!

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

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

The PrefixSpan Algorithm

- J. Pei, J. Han, et al. ICDE'01
- Notations/definitions: prefix and suffix (projection)
  - <a>, <aa>, <a(ab)> and <a(abc)> are *prefixes* of sequence <a(abc)(ac)d(cf)>
  - Given 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>

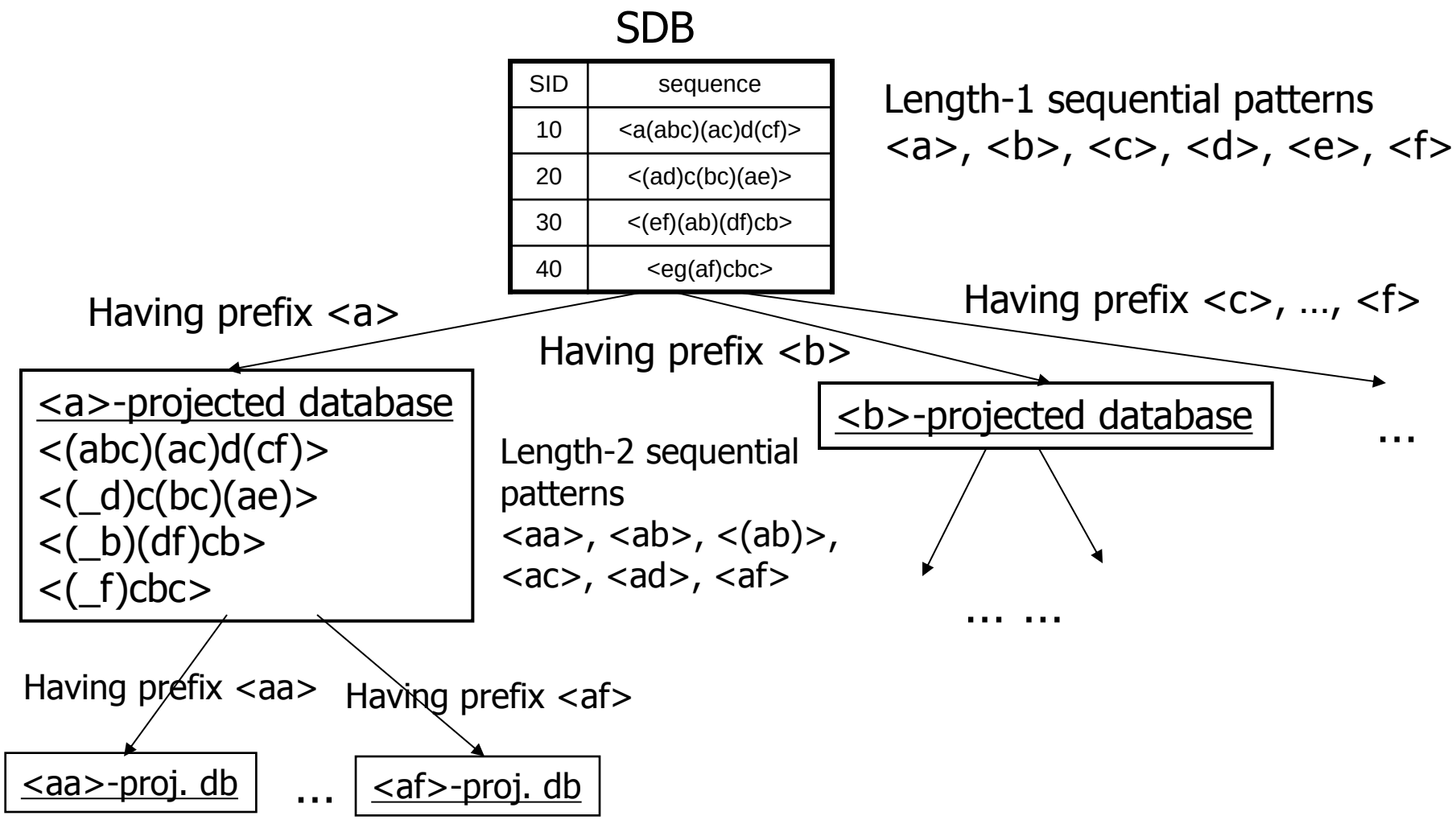
Finding Seq. Patterns with Prefix <a>

- 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

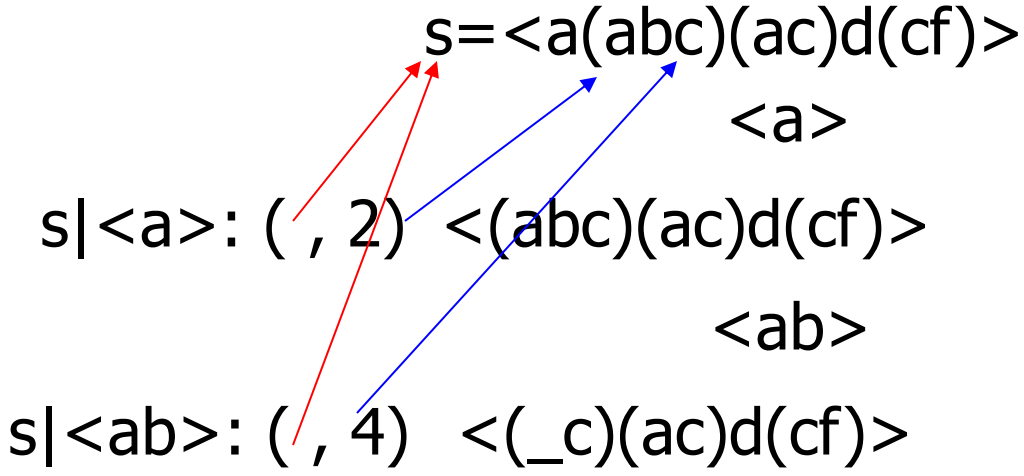


Efficiency of PrefixSpan

- 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

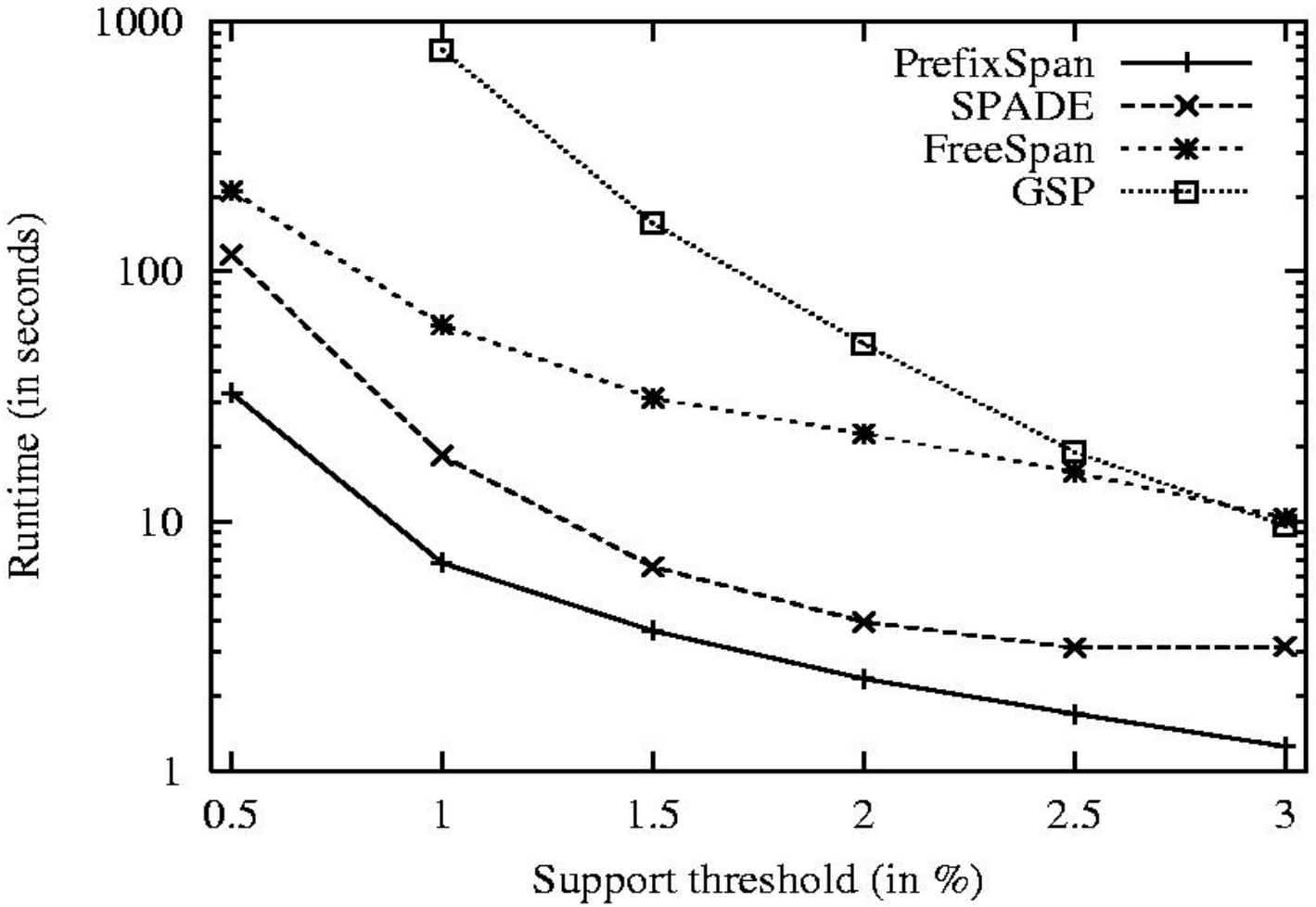
- Major cost of PrefixSpan: projection
  - Postfixes of sequences often appear repeatedly in recursive projected databases
- When (projected) database can be held in main memory, use pointers to form projections
  - Pointer to the sequence
  - Offset of the postfix



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



CloSpan: A Projection-based Closed Sequence Mining Algorithm

1	efabc
2	eabf
3	dcxa

- Mining closed sequences efficiently by prefix-pattern growth
- Example: a sequence database D
  - "e"-projected database (D|e):
    - fabc, abf.
  - "ea"-projected database (D|ea):
    - bc, bf
- Pseudo-projection optimization

CloSpan Optimization 1: Common Prefix Pruning

- If there exists a common prefix, all sequences beginning with a subsequence of this prefix should not be closed (except those beginning with this prefix)
- Example: abcd, abef, abde (min support is 2)
  - support(aβ) = support(abβ)
  - support(bβ) = support(abβ)
- Claim: aβ and bβ are not closed

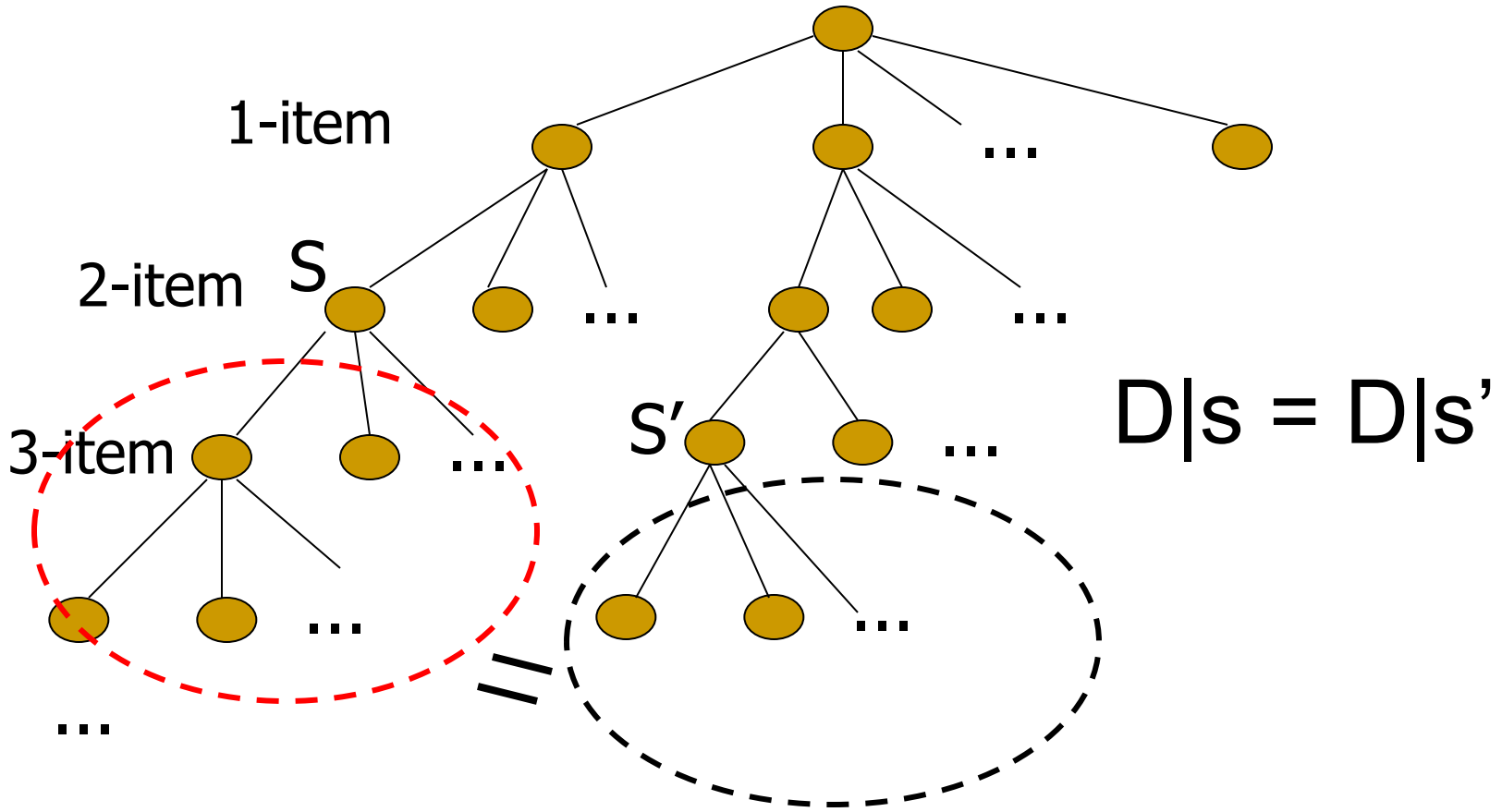
CloSpan Optimization 2: Partial-Order Pruning

- If “a” always occurs before “b” in all sequences, any sequence beginning with “b” will not be closed.
- Example: acbd, abed, adeb (min support is 2)
  - support(bβ) = support(abβ)
- Claim: bβ is not closed
- How about aβ?

CloSpan Optimization 3: Equivalent Projected DB Pruning

- Equivalence of Projected Database
  - Given s and s', s is a subsequence of s'
    - Size(D|s) = Size(D|s') ⟺ D|s = D|s'
- Example: abcdfg, bacfdgd, dcc
- Claim: D|acd = D|cd = "fg", "gd".
  - Size(D|acd) = Size(D|cd) ⇔ D|acd = D|cd
- Early Termination by Equivalence
  - Given s and s', s is a subsequence of s'
    - If Size(D|s) = Size(D|s')
    - Then no need to search the branch of s

Search Space Pruning in CloSpan



BIDE: Mining Closed Sequential Patterns without Candidate Maintenance-and-Test

- Efficient frequent sequence enumeration
  - Depth-first search, prefix-based pattern growth, pseudo-projection
- BI-Directional Extension closure checking scheme
  - Forward/backward-extension event checking
    - No need to maintain the set of already mined frequent closed sequences (or just candidates)
- Optimization techniques
  - BackScan search space pruning
  - ScanSkip optimization technique

Efficient Sequence Enumeration

- A divide-and-conquer approach
  - **f_list**: A:4, B:4, C:4
  - All seq. pat. can be divided into 3 subsets:
    - Seq. With prefix A
    - Those with prefix B
    - Those with prefix C

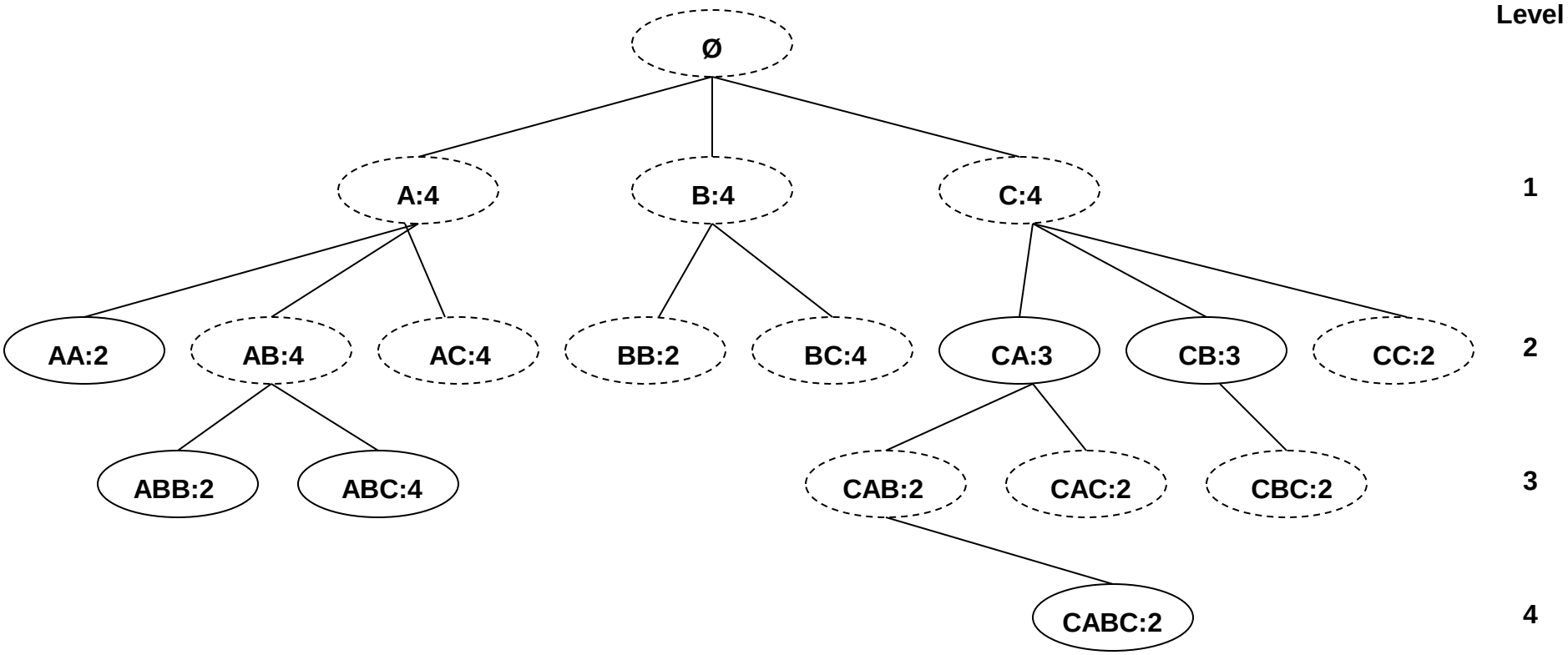
Sequence Database *SDB*

< C A A B C >
< A B C B >
< C A B C >
< A B B C A >

- Depth-first search of the sequence tree
  - Recursively mine seq. by extending a certain prefix according to certain item order (e.g., A ≤ B ≤C)
  - Running example (see the sequence tree)
    - Search order: A:4, AA:2, AB:4, ABB:2, ABC:4, AC:4, B:4, BB:2, BC:4, C:4, CA:3, CAB:2, CABC:2, CAC:2, CB:3, CBC:2, CC:2

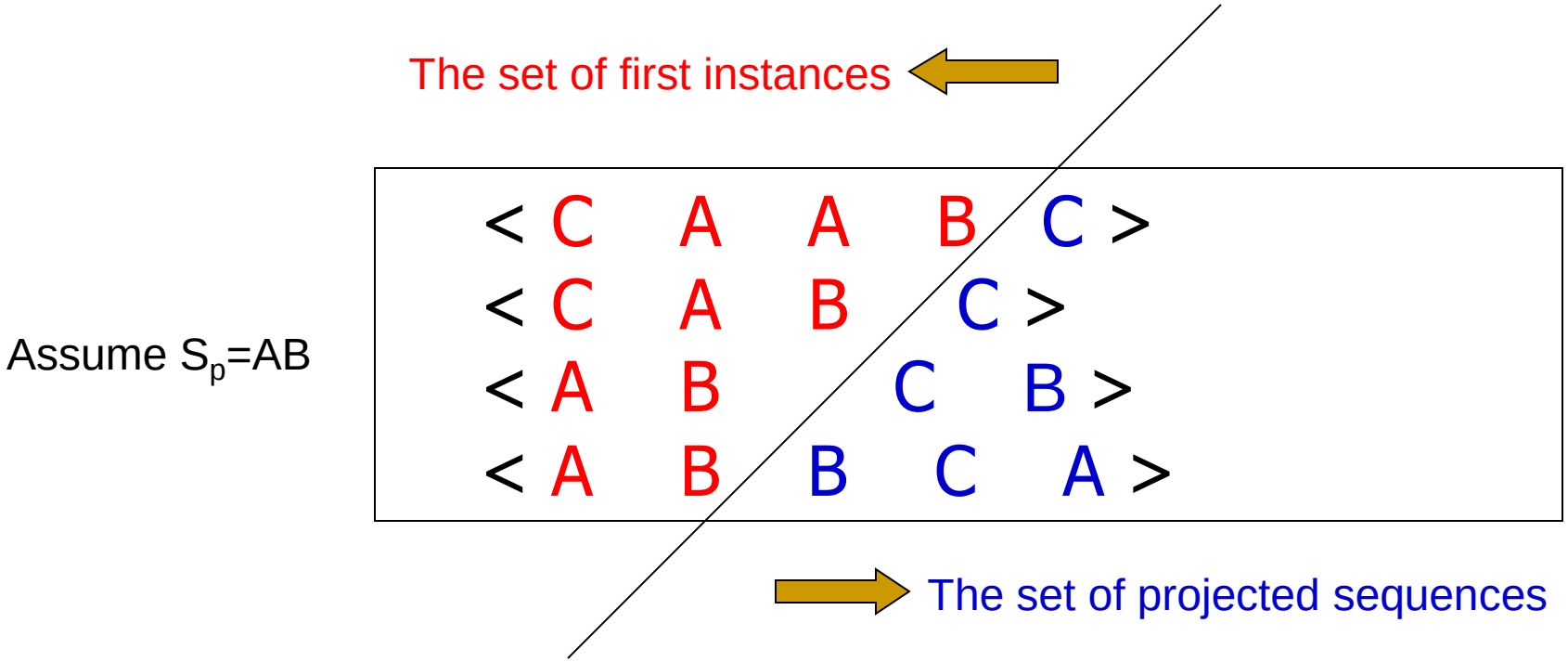
Search Space of Sequence Mining

- Assume item ordering: A ≤ B ≤ C, the complete search space of sequence mining forms a sequence tree



Prefix-based Projection

- First instance of a prefix S_p w.r.t. an input sequence S
  - From the beginning of S to the end of the first appearance of S_p
- Projected sequence of a prefix S_p w.r.t. an input sequence S
  - The remaining part of S after removing the 1st instance of S_p



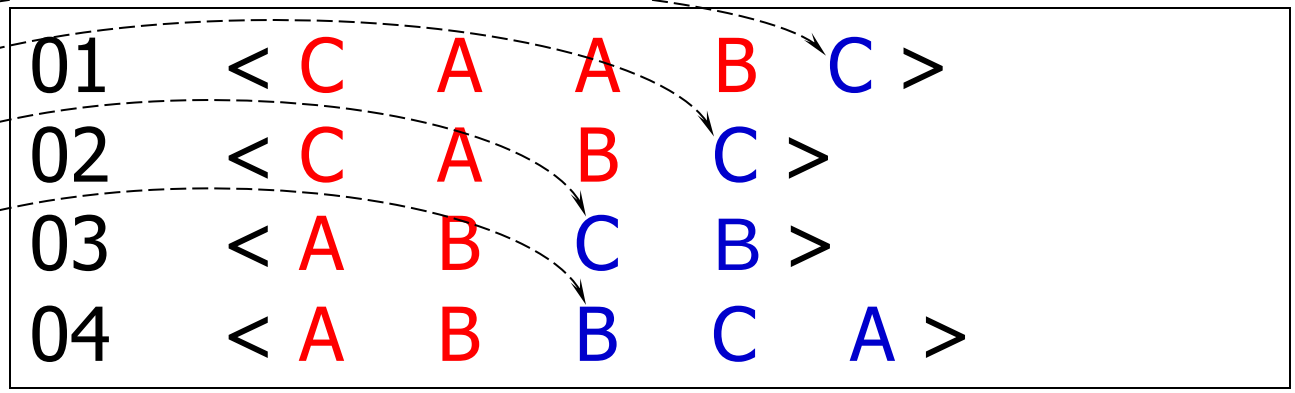
Pseudo Projection

- Physical projection is not space-efficient. Instead, use pointers to form pseudo projections
  - Pointer to the sequence (or sequence ID)
  - Offset of the projected sequence

Assume S_p=AB:

Pseudo-projections:

S-ID	offest
01	4
02	3
03	2
04	2



BI-Directional Extension: Closure Checking

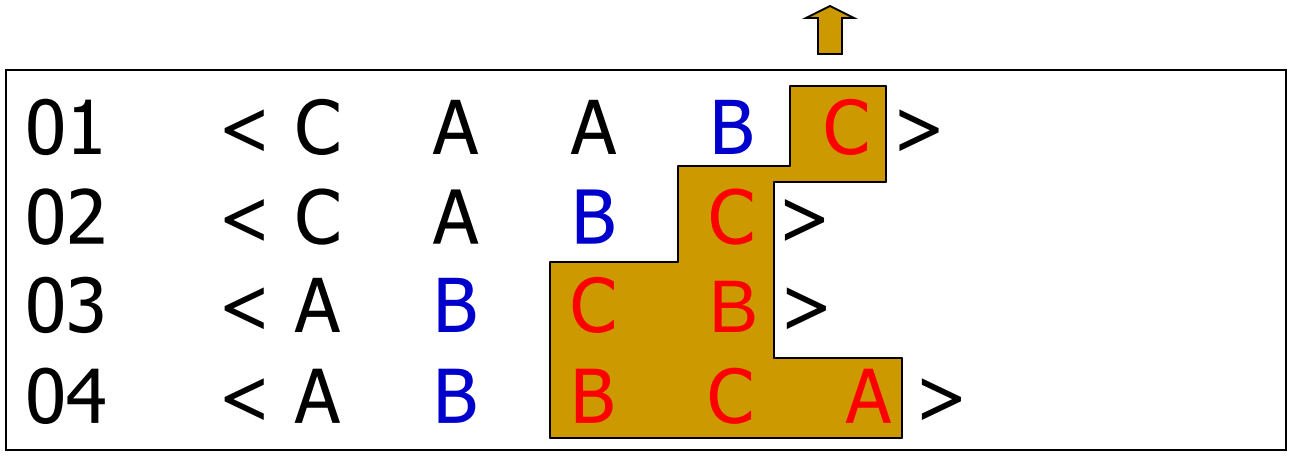
- BI-Directional Extension
  - Given a prefix sequence $S_p = e_1 e_2 \ldots e_n$
    - e' is called a **forward-extension event**, if $S_p' = e_1 e_2 \ldots e_n e'$ has the same support as S_p
    - e' is called a **backward-extension event**, if $S_p' = e_1 e_2 \ldots e' e_i \ldots e_n$ ($1 \leq i \leq n$) has the same support as S_p
- If there exists no *forward-extension event*, nor *backward-extension event* w.r.t. a prefix sequence S_p, S_p must be closed; otherwise, S_p must be non-closed
- How to check them efficiently ?

Forward-Extension Event Checking

- Given a prefix sequence $S_p = e_1 e_2 \ldots e_n$
  - The complete set of **forward-extension events** ≡ the set of its locally frequent items with the same support as S_p
  - e.g., S_p = AB, its locally frequent item set is {C:4, B:2}, C has the same support as S_p and thus is a forward-extension event

Assume S_p=AB:

Projected database

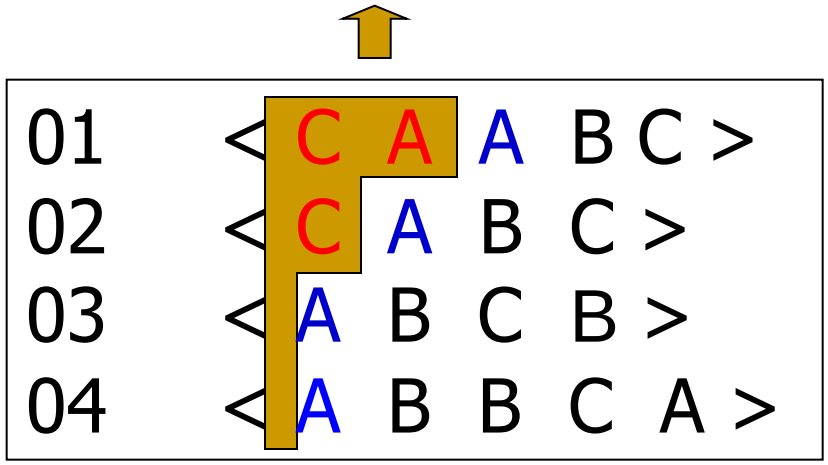


Backward-Extension Event Checking

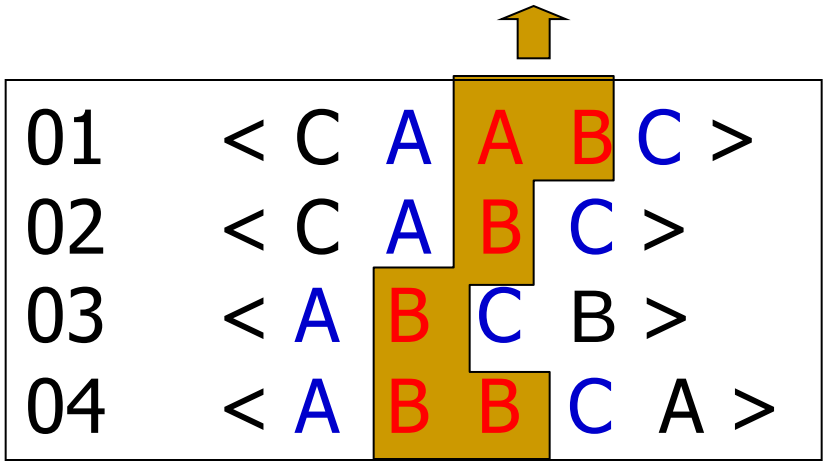
- Given a prefix sequence $S_p = e_1 e_2 \ldots e_n$ and an input sequence S
  - Check if there is any backward extension events in each of the i-th maximum period ($1 \le i \le n$)
  - i-th maximum period: the subsequence between the first instance of prefix $e_1 \ldots e_{i-1}$ and the last instance of suffix $e_i \ldots e_n$ *w.r.t. input sequence S*

Example: S_p=AC

No backward-extension event that appears in each of the 1st maximum period



B is a backward-extension event that appears in each of the 2nd maximum period

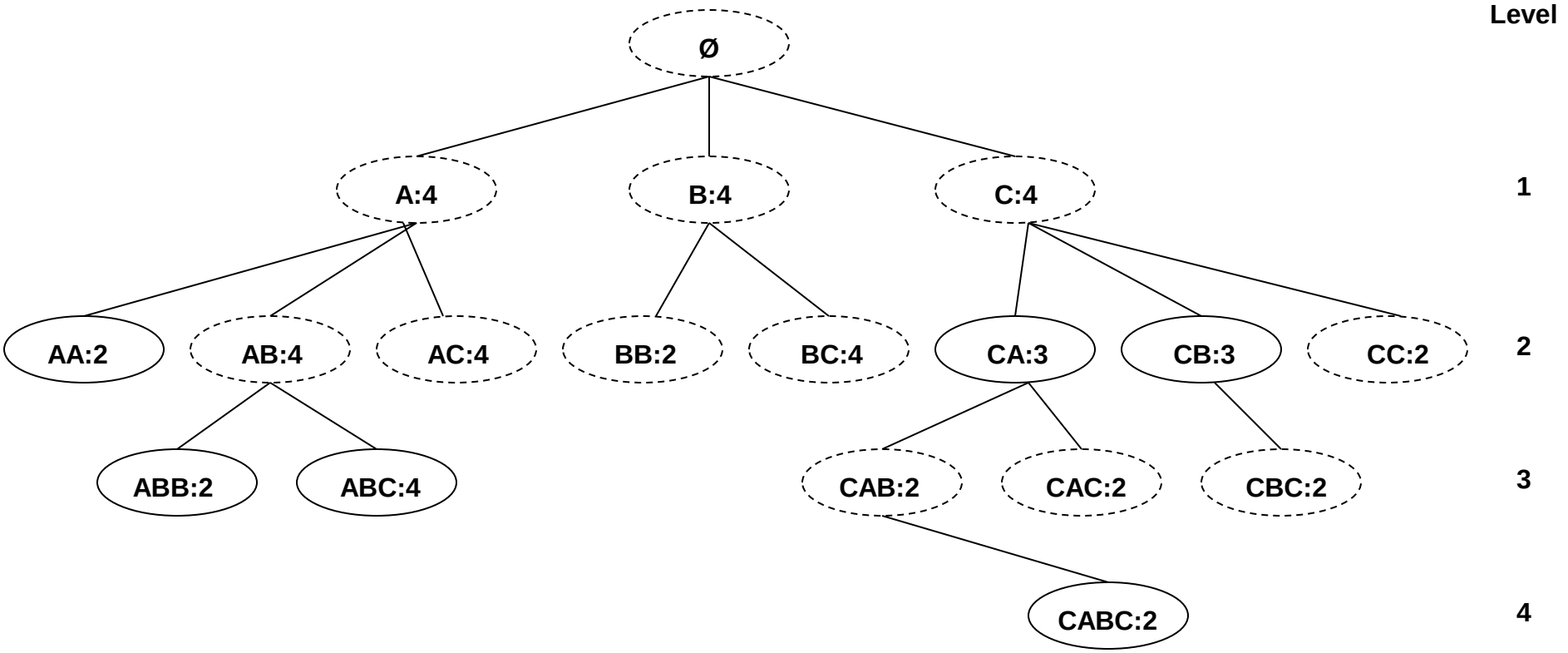


The Trick of Search Space Pruning

- Search space pruning is trickier in closed sequential pattern mining than closed itemset mining
- Example. Let A ≤ B ≤ C
  - The current prefix sequence is C:4
  - Can it be absorbed by an already mined sequence ABC: 4?
  - The answer is: no!
  - Because C can still generate CA:3, CABC:2, and CB:3
  - The temporal ordering makes things more complicated

Search Space of Sequence Mining

- Assume item ordering: A ≤ B ≤ C, the complete search space of sequence mining forms a sequence tree



BackScan Search Space Pruning

- Given a prefix sequence $S_p = e_1 e_2 \ldots e_n$ and an input sequence S
  - Check if there is any backward extension events in each of the i-th semi-maximum period ($1 \le i \le n$)
  - i-th semi-maximum period: the subsequence between the first instance of prefix $e_1 \ldots e_{i-1}$ and the last instance of suffix $e_i \ldots e_n$ *within the first instance of* S_p w.r.t. S

Example: S_p=B:4 can be pruned

A is a backward-extension event that appears in each of the 1st semi-maximum period

Set of 1st semi-maximum periods

01	< C A A B C >
02	< C A B C >
03	< A B C B >
04	< A B B C A >

Example: S_p=C:4 cannot be pruned

No backward-extension event that appears in each of the 1st semi-maximum period

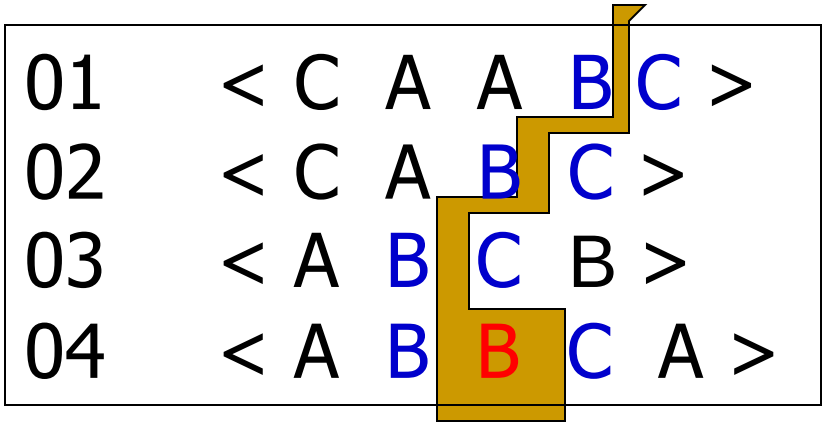
01	< C A A B C >
02	< C A B C >
03	< A B C B >
04	< A B B C A >

Set of 1st semi-maximum periods

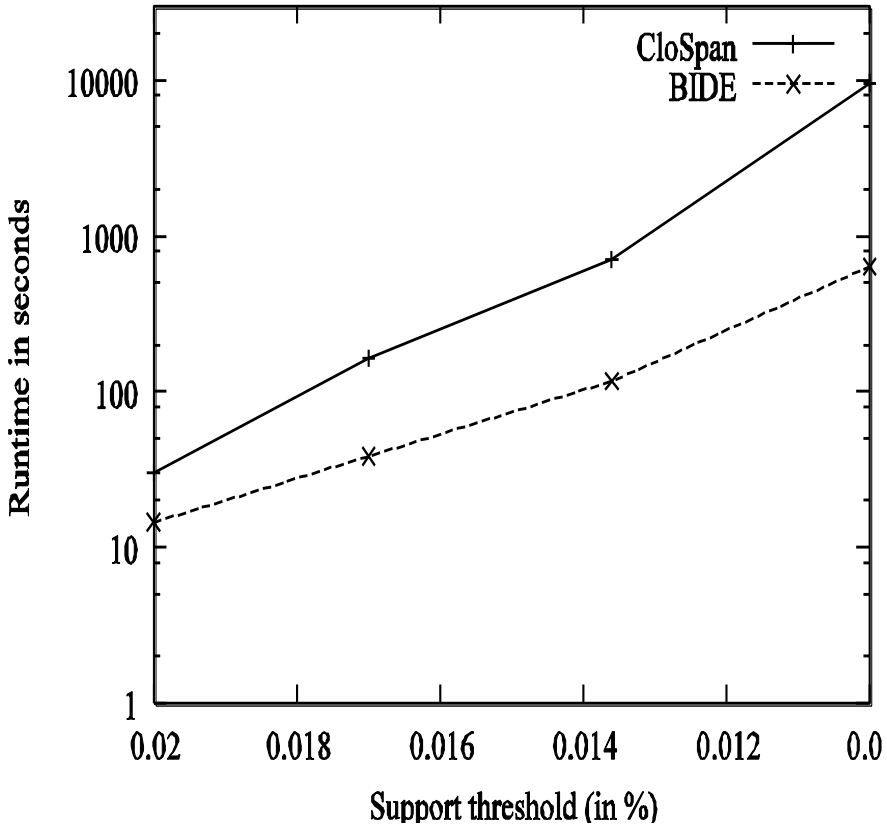
ScanSkip Optimization Technique

- Given a prefix sequence $S_p = e_1 e_2 \ldots e_n$ and an input sequence S
  - In many cases we can skip the scanning of some of the maximum periods (in BI-Directional Extension closure checking) or the semi-maximum periods (in BackScan pruning)

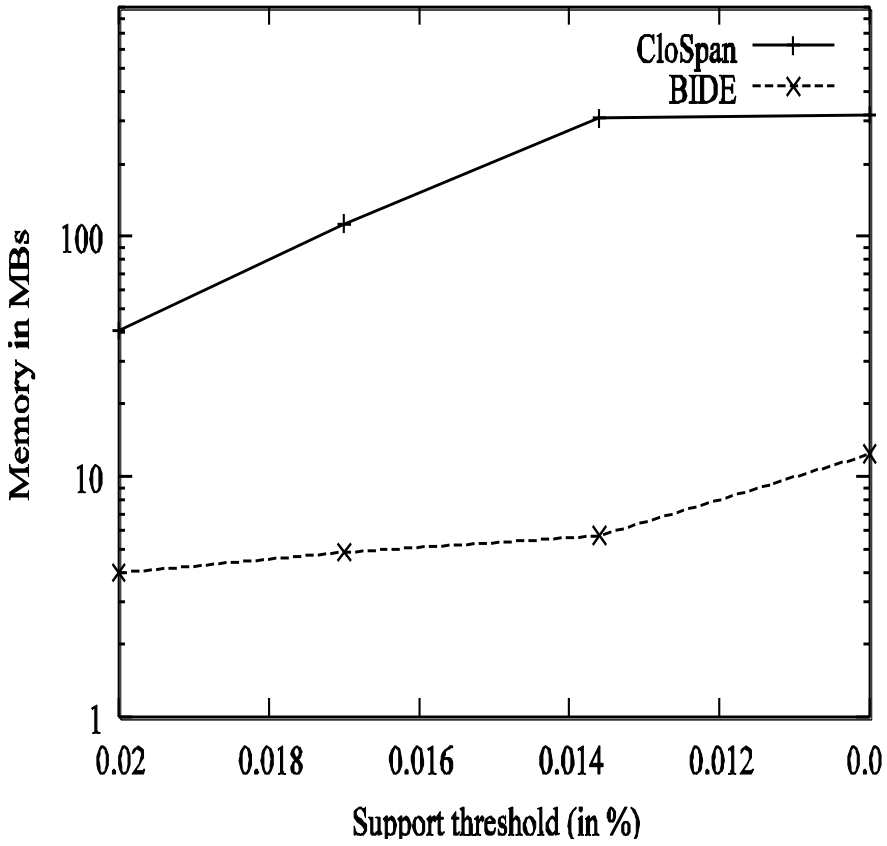
Example: S_p = ABC:4, the 3rd maximum periods w.r.t. ABC:4 is { ø, ø, ø, B}



Performance Evaluation: BIDE vs. CloSpan

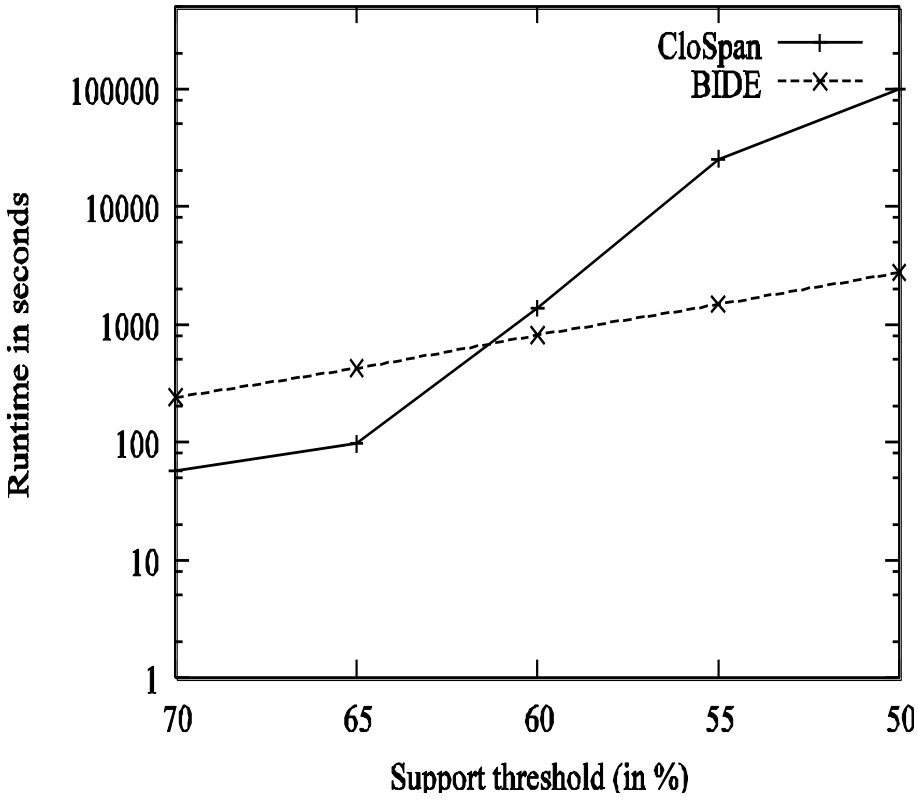


Runtime comparison, Gazelle dataset

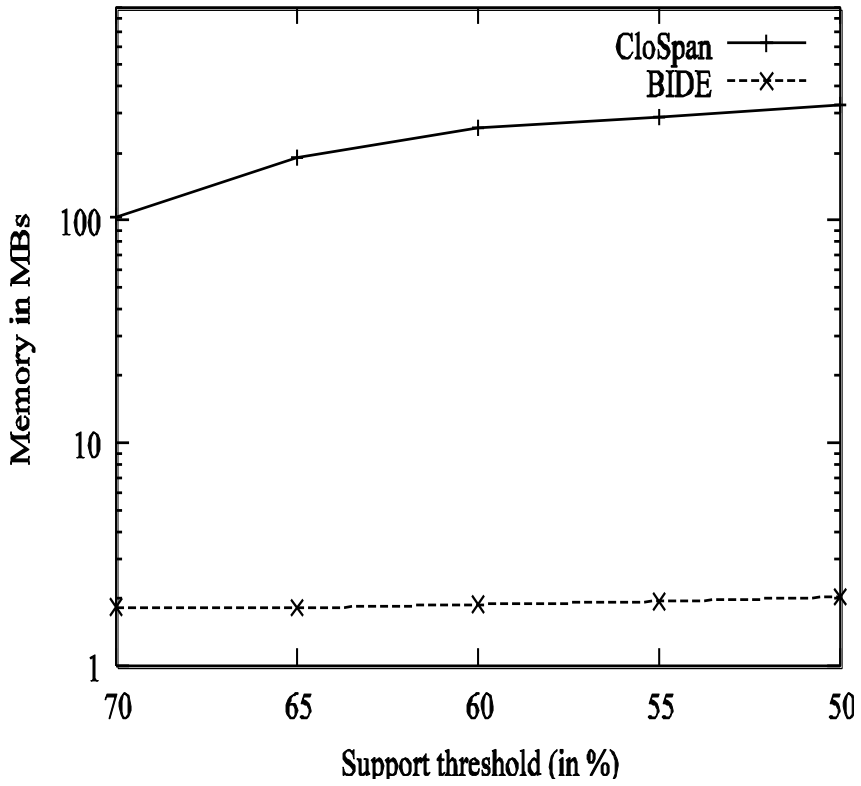


Memory comparison, Gazelle dataset

Performance Evaluation: BIDE vs. CloSpan

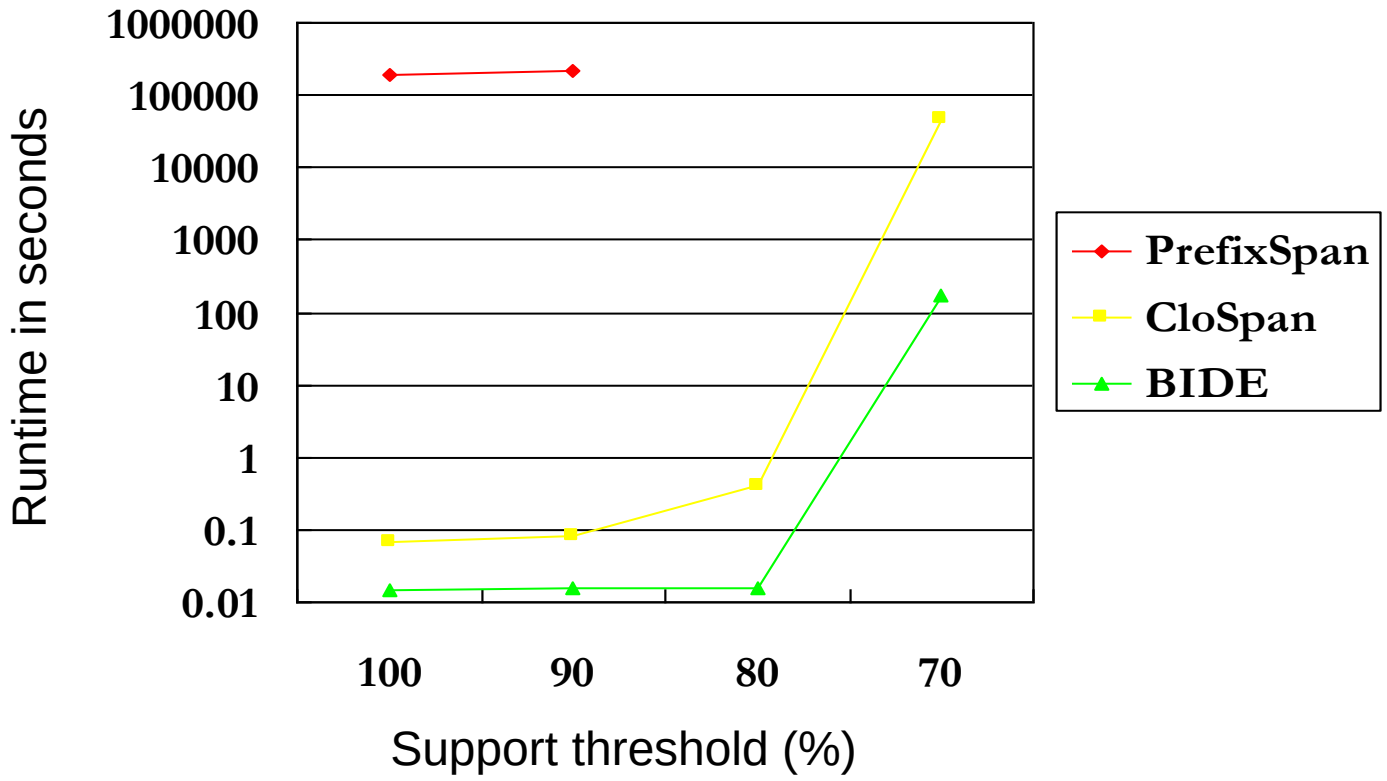


Runtime comparison, snake dataset



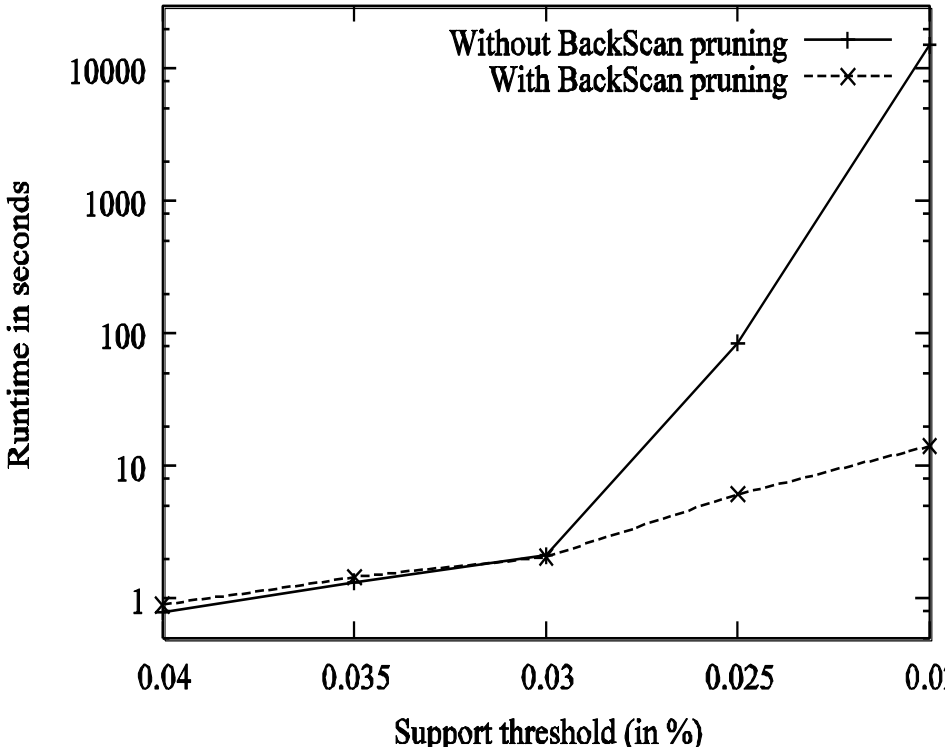
Memory comparison, snake dataset

Performance Evaluation: BIDE vs. CloSpan

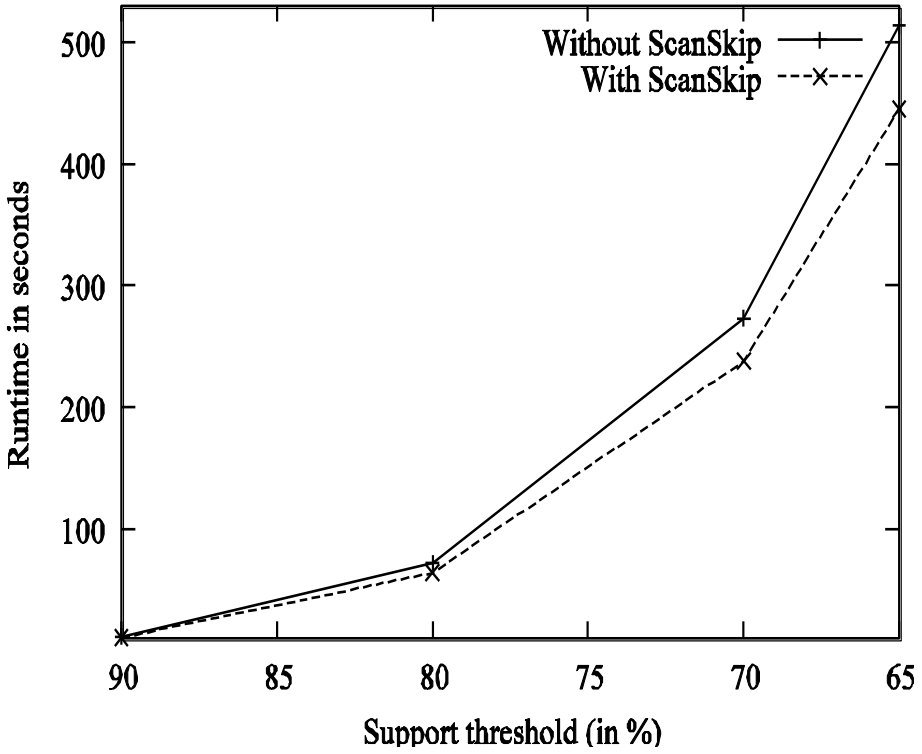


Runtime comparison, Program-trace dataset

Optimization Technique Evaluation in BIDE

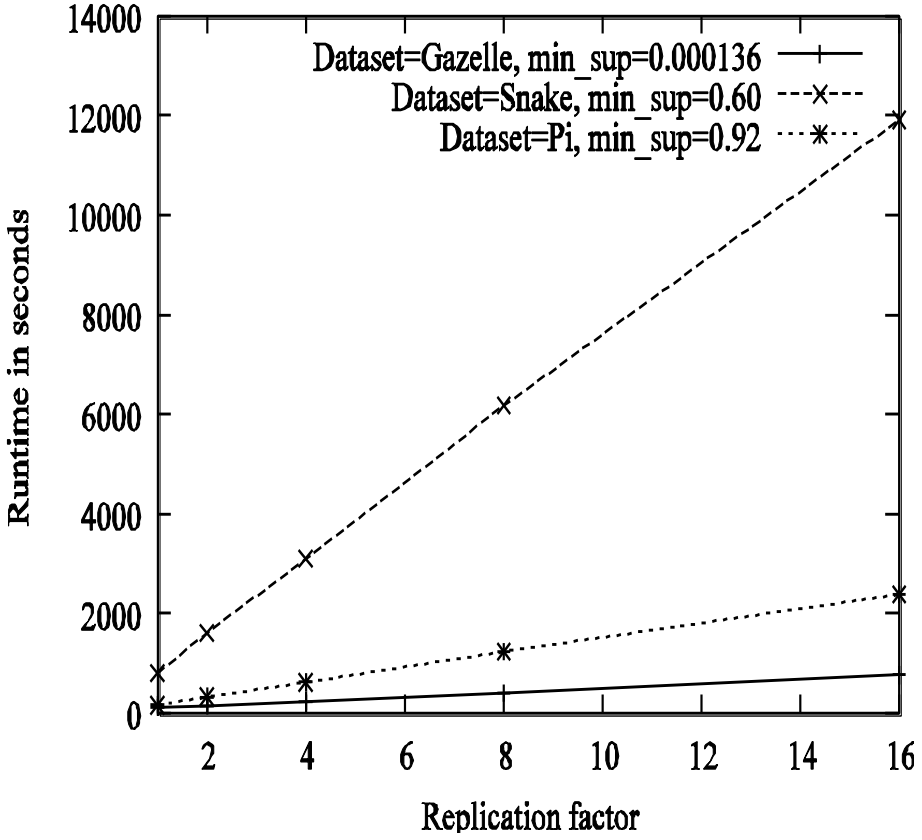


BackScan pruning, Gazelle dataset

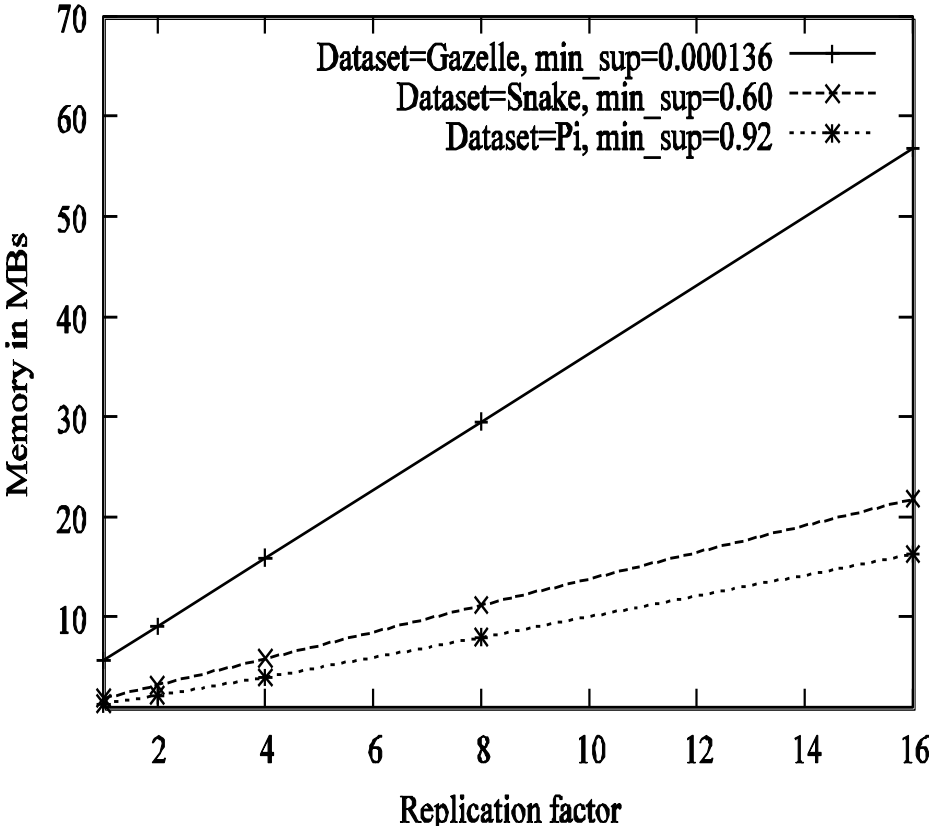


ScanSkip optimization, snake dataset

Scalability Test of BIDE



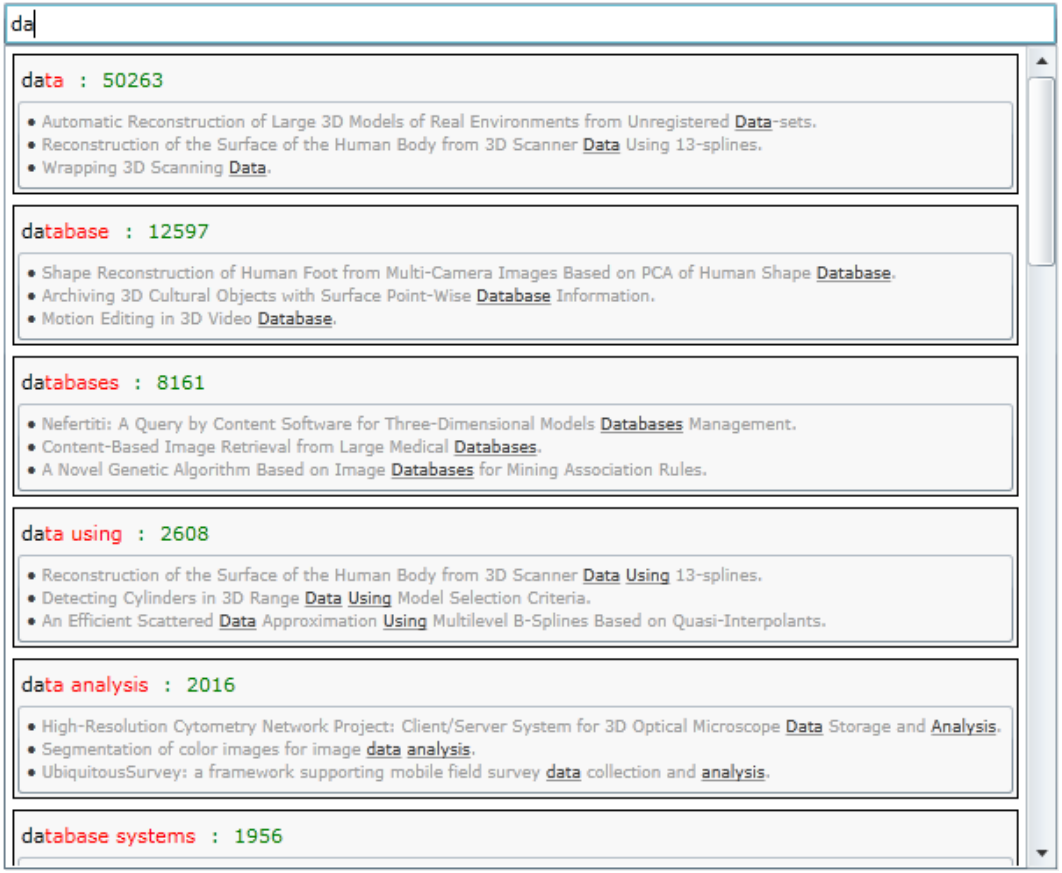
Runtime



Memory

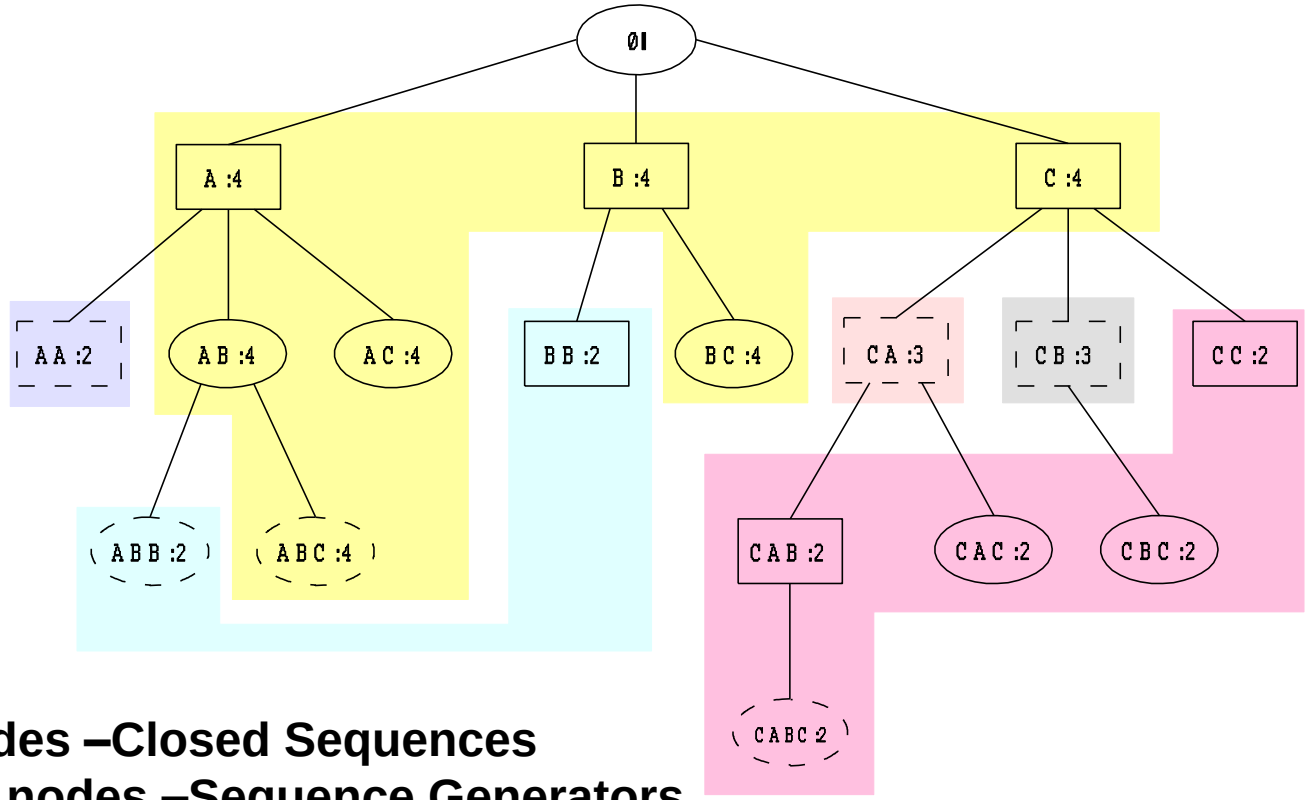
One Application of Closed Sequential Pattern Mining

- Sequential pattern based query completion on multi-column structural data
  - ✓ ACM CIKM'10(demo)



Sequence Genearor Mining

- Effcient mining of frequent sequence generators:
  - C. Gao, J. Wang, Y. He, L. Zhou. WWW'08.



Dotted nodes –Closed Sequences
Rectangle nodes –Sequence Generators

Sequence Generator Mining

- Application 1
  - Opinion mining/Sentiment analysis/Review Categorization
    - » An example: "<trying, how> ➔Negative" is a rule learnt from the Office07Review dataset
      - ∴ I'm still trying to figure out how to type in Chinese ...
- Application 2
  - Erroneous sentence detection
    - » An example: "<this, NNS>➔Non-native" is a rule learnt from the ESL-Chinese dataset
      - ∴ In all this skills are make us dazzled.

Conclusions

- Sequential pattern mining
  - GSP, SPADE, PrefixSpan, SPAM
- Closed subsequence mining
  - CloSpan, BIDE
  - BIDE: An efficient closed sequential pattern mining algorithm
    - **BI-Directional Extension** closure checking
    - **BackScan** search space pruning
    - **ScanSkip** optimization technique
- Sequential pattern mining is still an active research area

Thanks!