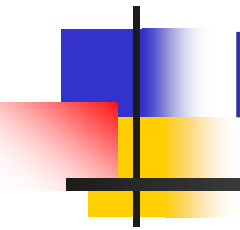
Graph and Web Mining - Motivation, Applications and Algorithms

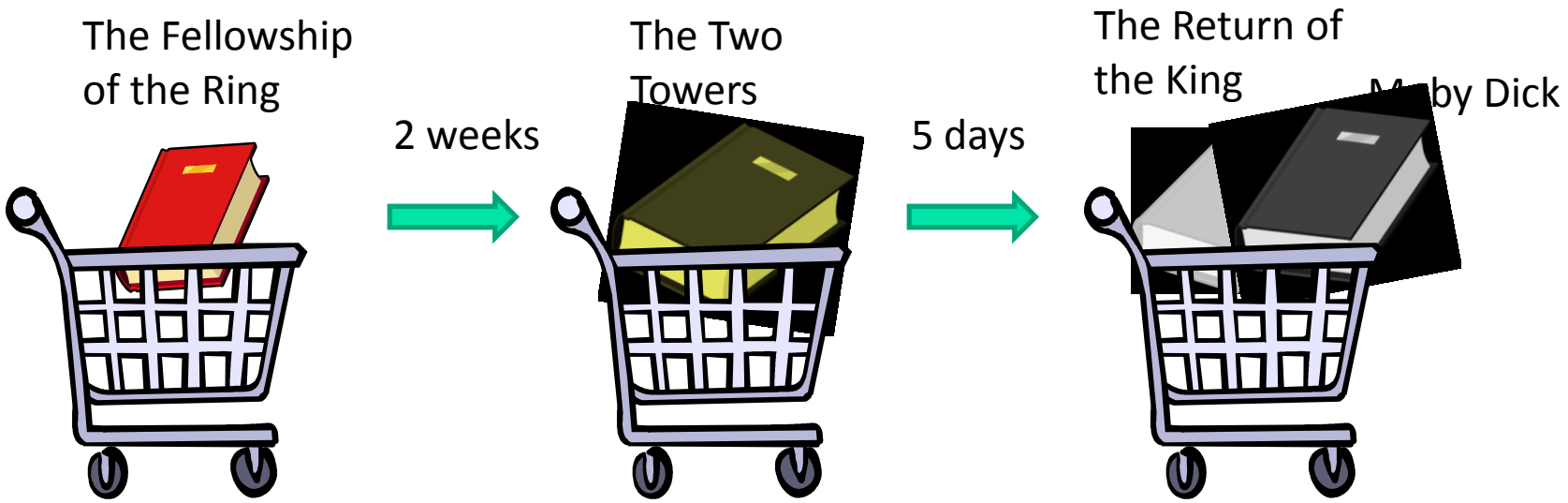
Prof. Ehud Gudes
Department of Computer Science
Ben-Gurion University, Israel



Finding Sequential Patterns

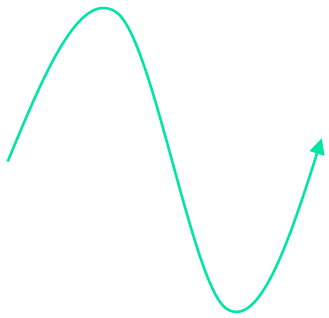
Sequential Patterns Mining

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



More Detailed Example

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



Min Support = 0.5

Frequent Sequences
<a>
<(a)(a)>
<(a)(c)>
<(a)(bc)>
<(e)(a)(c)>
...

Motivation

- Business:
  - Customer shopping patte
  - telephone calling patterns
  - Stock market fluctuation
  - Weblog click stream analysis
- Medical Domains:
  - Symptoms of a diseases
  - DNA sequence analysis

Definitions

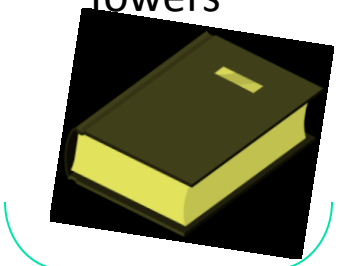
- **Items**: a set of literals $\{i_1, i_2, \ldots, i_m\}$
- **Itemset** (or event): a non-empty set of items.
- **Sequence**: an ordered list of itemsets, denoted as <(abc)(aef)(b)>
- A sequence $<a_1 \ldots a_n>$ is a **subsequence** of sequence $<b_1 \ldots b_m>$ if there exists integers $i_1 < \ldots < i_n$ such that $a_1 \subseteq b_{i_1}, \ldots, a_n \subseteq b_{i_n}$

Definitions

The Fellowship of the Ring

2 weeks

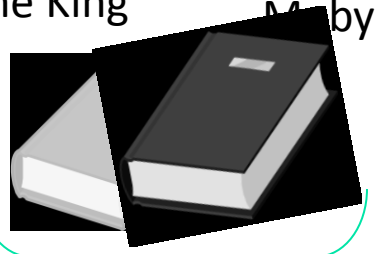
The Two Towers



5 days

The Return of the King

Moby Dick

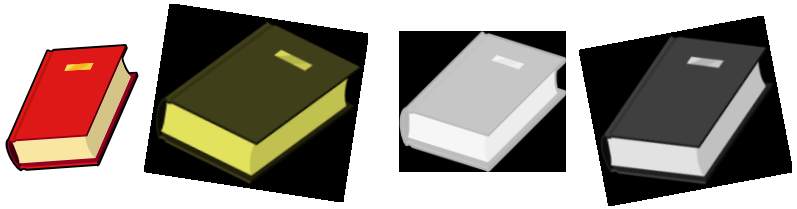


event

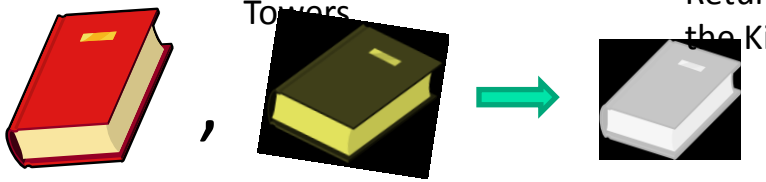
event

event

Items:



subsequences:



The Two Towers

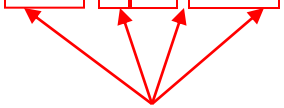
The Return of the King

Definitions

A *sequence database*

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)>

A *sequence* : <(bd)c b (ac)>



Events

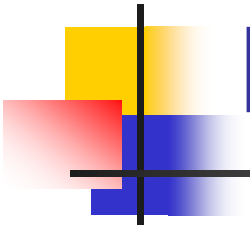
<ad(ae)> is a *subsequence* of <a(bd)bcb(ade)>

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

Much Harder than Frequent Itemsets!

2^{m*n} possible candidates!

Where *m* is the number of items, and *n* in the number of transactions in the longest sequence.



More Definitions

- *Support* is the number of **sequences** that contain the pattern. (as in frequent itemsets, the concept of *confidence* is **not** defined)

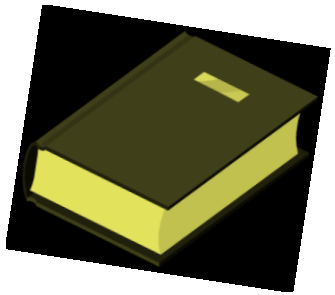
More Definitions

- **Min/Max Gap**: maximum and/or minimum time gaps between adjacent elements.

The Fellowship of the Ring

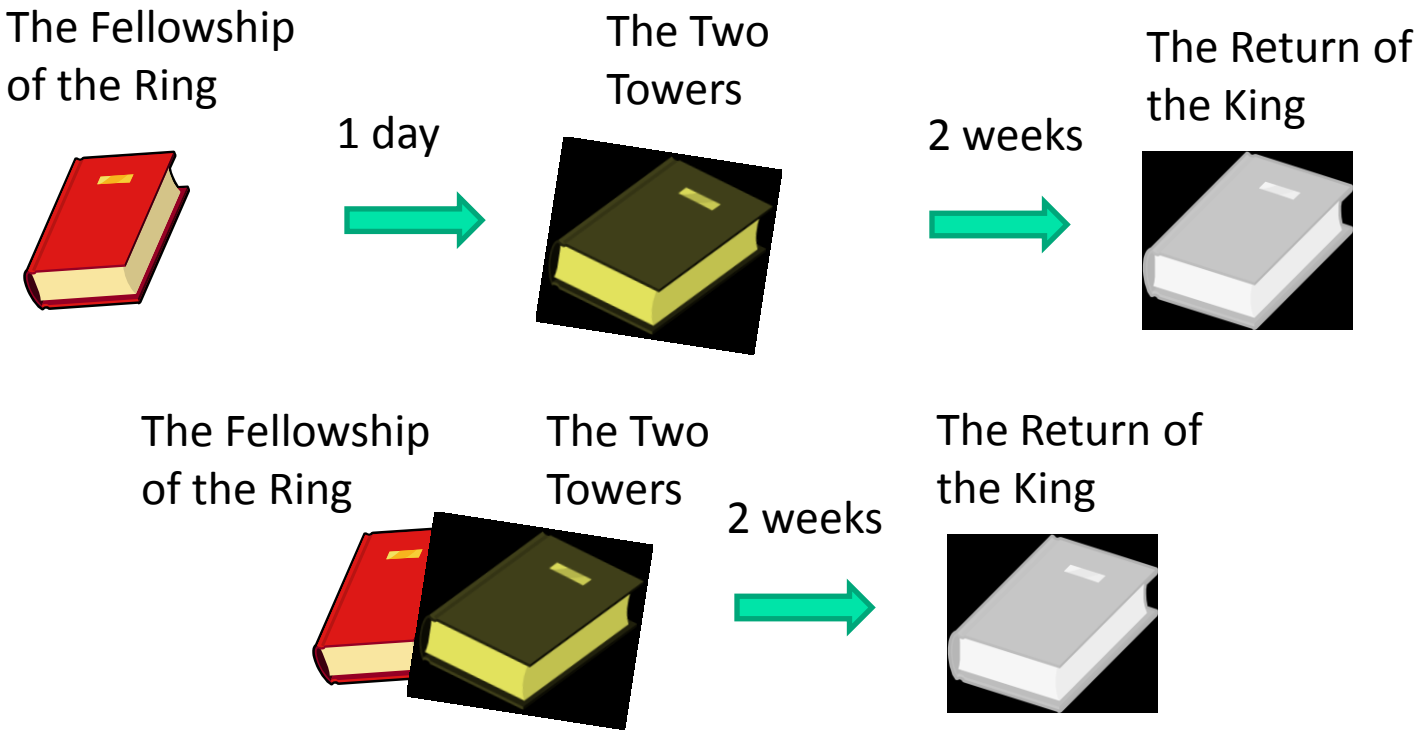
3 years

The Two Towers



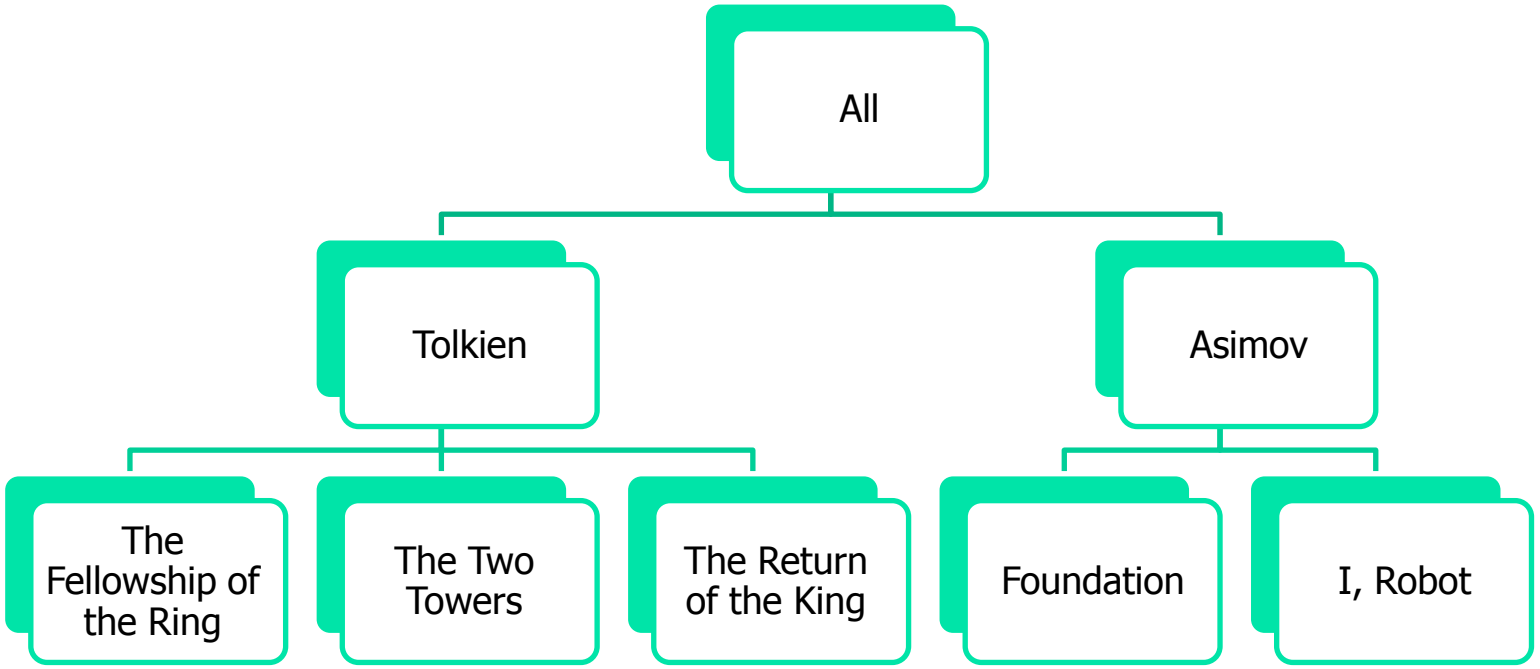
More Definitions

- **Sliding Windows**: consider two transactions as one as long as they are in the same time-windows.



More Definitions

- **Multilevel**: patterns that include items across different levels of hierarchy.



More Definitions

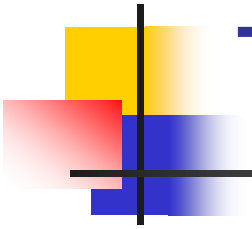
- Multilevel

Tolkien

Tolkien

The Return of the King

The GSP Algorithm



- Developed by Srikant and Agrawal in 1996.
- Multiple-pass over the database.
- Uses generate-and-test approach.

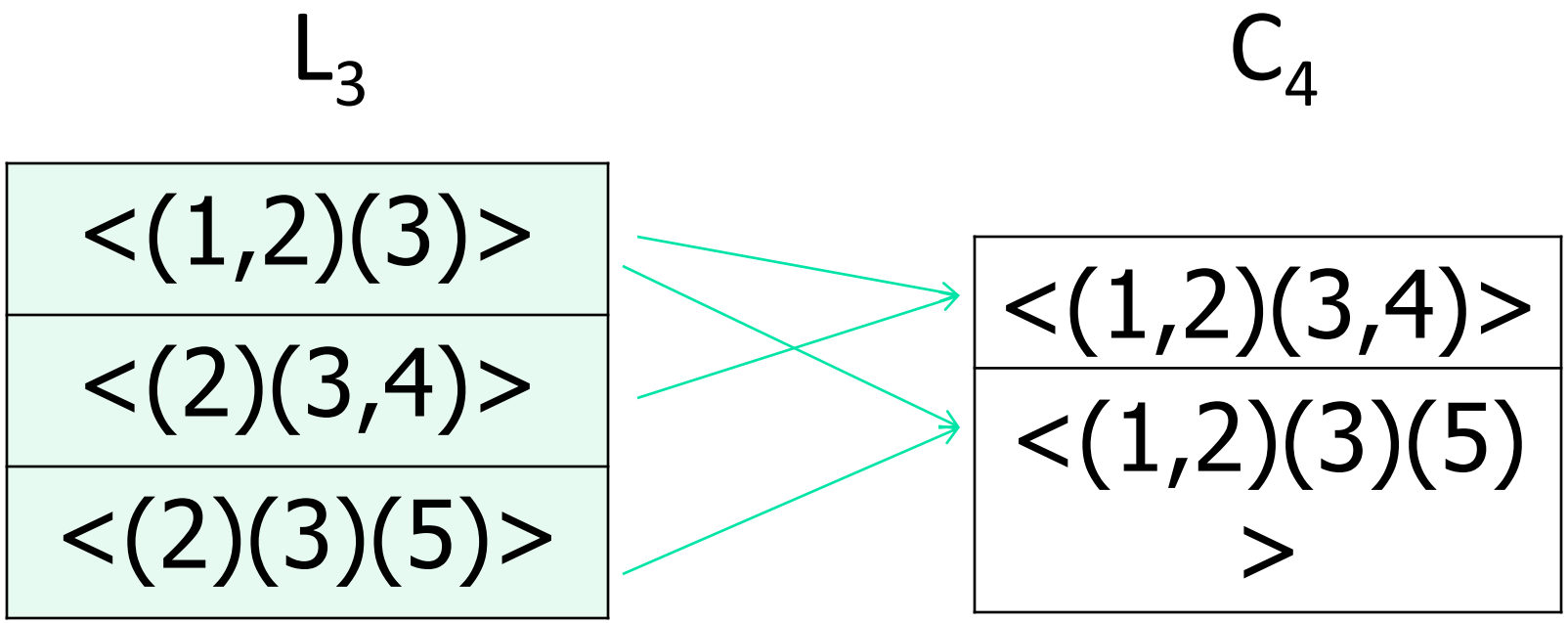
The GSP Algorithm

- **Phase 1**: makes the first pass over database
  - To yield all the 1-element frequent sequences. Denoted L_1.
- **Phase 2**: the Kth pass:
  - starts with seed set found in the (k-1)th pass (L_{k-1}) to generate candidate sequences, which have one more item than a seed sequence; denoted C_k.
  - A new pass over *D* to find the support for these candidate sequences
- **Phase 3**: Terminates when no more frequent sequences are found

The GSP Algorithm Candidate Generation

- Joining L_{k-1} with L_{k-1}: a sequence s_1 joins with s_2 if dropping the first item from s_1 and dropping the last item from s_2 makes the same sequence.
- The added item becomes a separate event if it was a separate event in s_2, and part of the last event in s_1 otherwise.
- When joining L_1 with L_1 we need to add both ways.

Candidate Generation Example



Example

Min support =50%

DB

SID	sequence
1	<a(abc)(ac)d(cf)>
2	<(ad)c(bc)(ae)>
3	<(ef)(ab)(df)cb>
4	<eg(af)cbc>

C_1

SEQ	Sup
<a>	4
<b>	4
<c>	3
<d>	3
<e>	3
<f>	3
<g>	1

→ L_1

SEQ
<a>
<b>
<c>
<d>
<e>
<f>

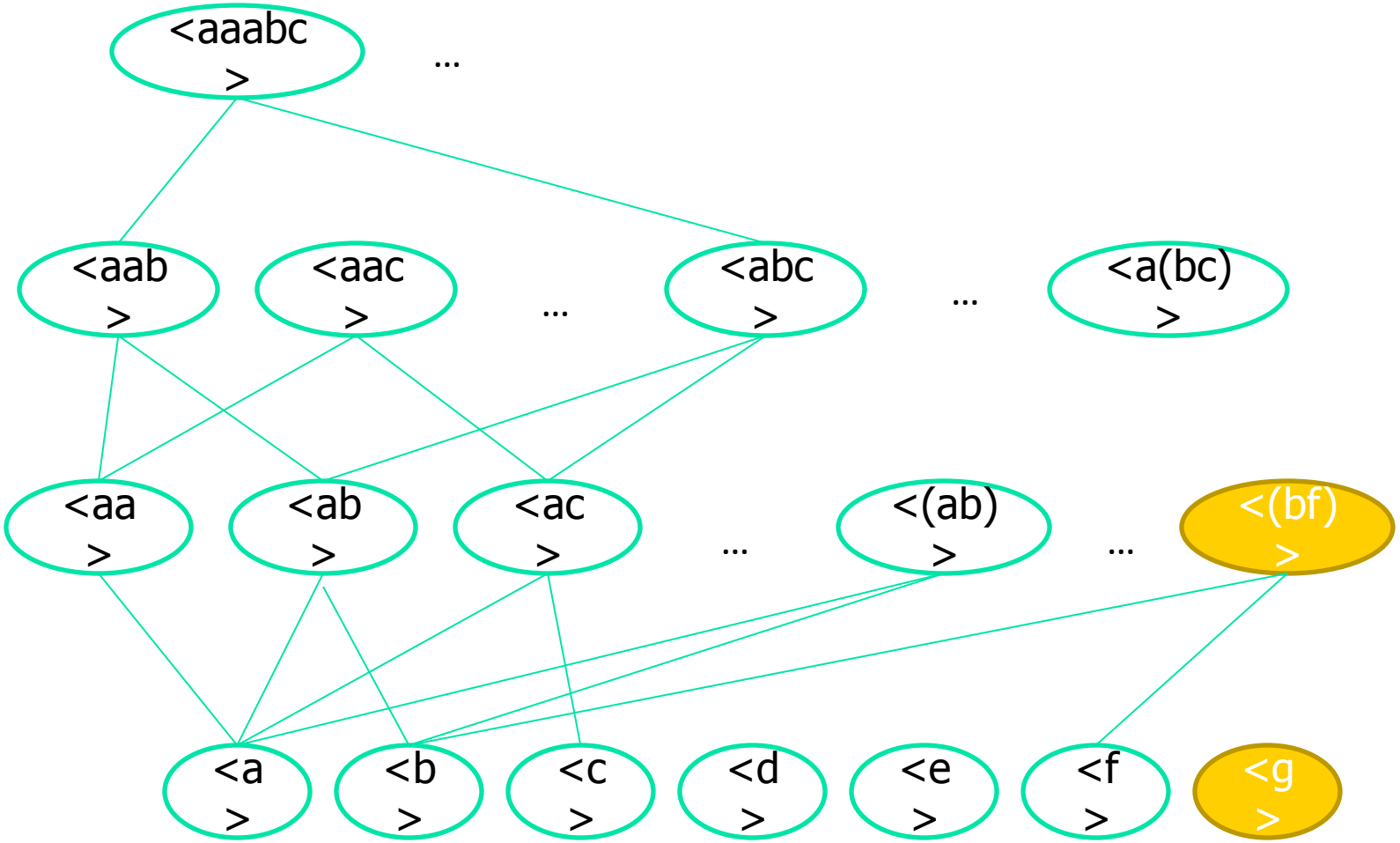
→ L_1 x L_1

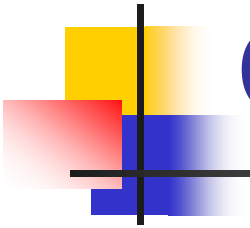
C_2

SEQ	Sup
<aa>	2
<ab>	4
…	
<af>	2
<ba>	2
<bb>	1
…	
<ff>	0
<(ab)>	2
<(ac)>	1
…	
<(ef)>	0

$$6 \times 6 + \frac{6 \times 5}{2} = 51$$

Same Example – Lattice Look





GSP 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** (**S**equential **PA**ttern **D**iscovery using **E**quivalent 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

SPADE: How It Works

Horizontal

SID	sequence
1	<a(abc)(ac)d(cf)>
2	<(ad)c(bc)(ae)>
3	<(ef)(ab)(df)cb>
4	<eg(af)cbc>

Vertical

SID	EID	itemset
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
…	…	…
4	6	c

SPADE: How It Works

ID Lists for some 1-sequence

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			

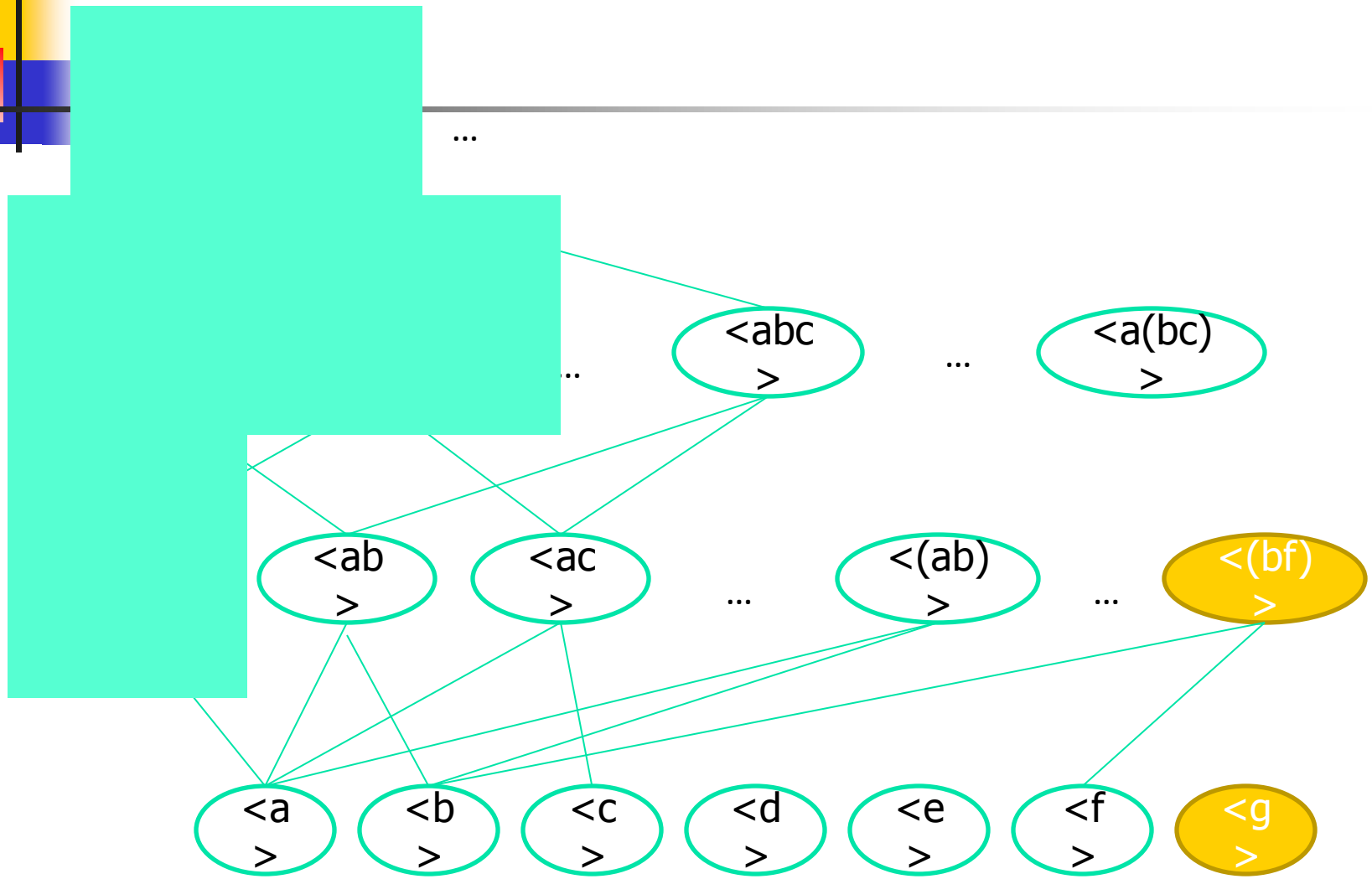
ID Lists for some 2-sequence

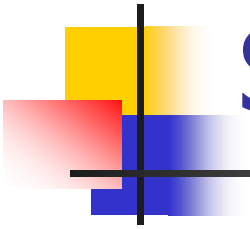
ab			ba			...
SID	EID(a)	EID(b)	SID	EID(b)	EID(a)	
1	1	2	1	2	3	
2	1	3	2	3	4	
3	2	5				
4	3	5				

ID Lists for some 3-sequence

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

SPADE: Equivalence Class

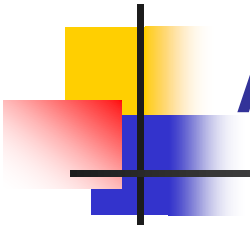




SPADE: Conclusion

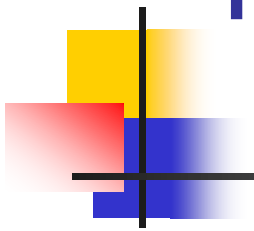
- The ID Lists carry the information necessary to find support of candidates. Reduces scans of the sequence database.
- However, basic methodology is breadth-first search and pruning, like GSP.

Pattern Growth: A Different Approach - PrefixSpan



- Does not require candidate generation.
- General Idea:
  - Find frequent single items.
  - Compress this information into a tree.
  - Use tree to generate a set of **projected databases**.
  - Each of these databases is mined separately.

Prefix and Suffix (Projection)



- Let s=<a(abc)(ac)d(cf)>
- <a>, <aa> and <a(ab)> are prefixes of s.

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

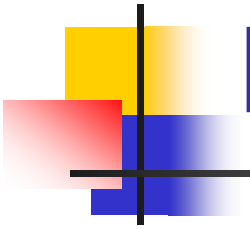
- Step 1: find length-1 sequential patterns
  - <a>, <b>, <c>, <d>, <e>, <f>
- Step 2: divide search space. The complete set of seq. pat. can be partitioned into 6 subsets:
  - The ones having prefix <a>;
  - The ones having prefix <b>;
  - …
  - The ones having prefix <f>

SID	sequence
1	<a(abc)(ac)d(cf)>
2	<(ad)c(bc)(ae)>
3	<(ef)(ab)(df)cb>
4	<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
1	<a(abc)(ac)d(cf)>
2	<(ad)c(bc)(ae)>
3	<(ef)(ab)(df)cb>
4	<eg(af)cbc>



Efficiency of PrefixSpan

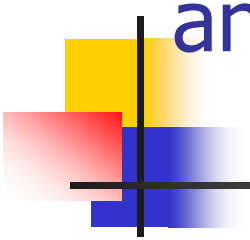
- No candidate sequence needs to be generated
- Projected databases keep shrinking
- Major cost of PrefixSpan: constructing projected databases
- Found to be more efficient than Spade

Constraint-Based Sequential Pattern Mining

- Constraint-based sequential pattern mining
  - Constraints: User-specified, for focused mining of desired patterns
  - How to explore efficient mining with constraints? — Optimization

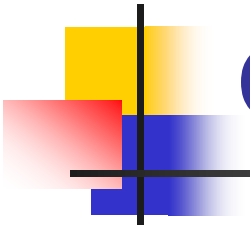
- Classification of constraints
  - **Anti-monotone**: E.g., sum(S) < 150 (If S doesn't fulfill the constraint so will super_sequence of S)
  - **Monotone**: E.g., count (S) > 5 (If S does fulfill the constraint so will super_sequence of S)
  - **Succinct**: E.g., length(S) ≥ 10, S ? (the set of sequences fullfilling the constrained can be defined precisely)
  - **Time-dependent**: E.g., min gap, max gap, total time.

Problems with Current approaches – Spade and PrefixSpan (and their variations)



- Fail (don’t terminate) on database with long sequences
- Do not handle efficiently the various constraints

Our ideas - SPADE Improvement + Constraints



- Use the vertical data format
- Two phase algorithm:
  - Frequent itemset phase
    - Use the well-knows Apriori Algorithm to mine frequent itemsets.
    - Apply itemset constraints: max itemset length, items that cannot occur together.
  - Sequence phase
    - Apply sequence constraints: max gap, min gap, max/min sequence length.

Result: the CAMLS Algorithm

Frequent Itemset Phase

- Use Apriori or FP-Growth to find frequent itemsets.

SID	sequence
1	<a(abc)(ac)d(cf)>
2	<(ad)c(bc)(ae)>
3	<(ef)(ab)(df)cb>
4	<eg(af)cbc>

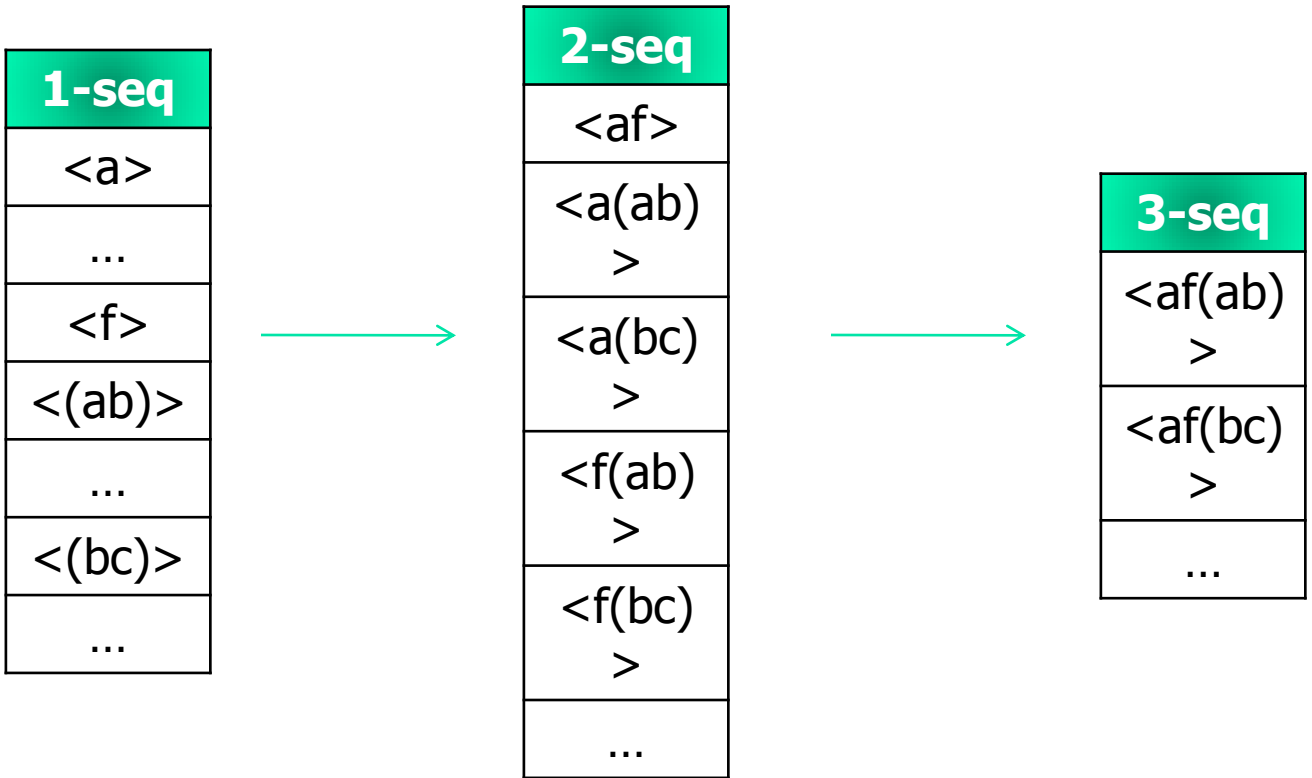
SID	itemset
1	a
1	abc
1	ac
1	d
1	cf
2	ad
2	c
…	…
4	c

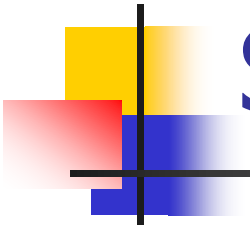
Frequent Itemsets

itemset
a
…
f
ab
…
bc
…

Sequence Phase

- Similar to GSP's and SPADE's candidate generation phase – except using the frequent itemsets as seeds



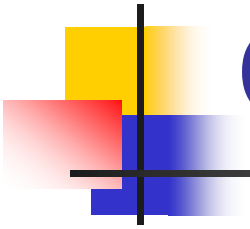


So What do We Get?

- The best of both worlds:
  - Much less candidates are being generated.
  - Support check is fast.
  - Worst case: works like SPADE.
  - Tradeoff: Uses a bit more memory (for storing the frequent item-sets).

CAMLS

- **C**onstraint-based **A**priori algorithm for **M**ining **L**ong **S**equences
- Designed especially for efficient mining of long sequences
- Uses constraints to increase efficiency
- Outperforms both SPADE and Prefix Span on both synthetic and real data
- Appeared in DASFAA 2010



CAMLS

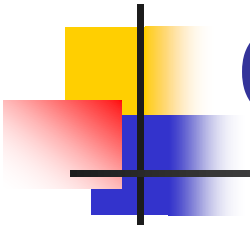
Makes a logical distinction between two types of constraints:

- Intra-Event: constraints that are not time related (such as items), e.g.: *Singletons*
- Inter-Event: temporal aspect of the data, i.e. values that can or cannot appear one after the other sequentially, e.g.: *Maxgap*
- Use an innovative pruning strategy

Tested Domain – predicting Machine failures

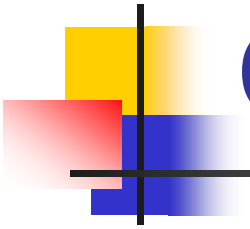
Quartz-Tungsten-Halogen lamp - used in the semiconductors industry for finding defects in a chip manufacturing process.

- Long sequences
- Restricted number of items in event
- Relatively small number of frequent events



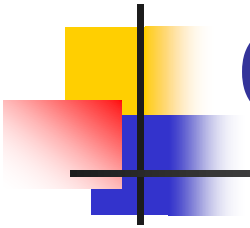
CAMLS

- Consists of two phases corresponding to the two types of constraints:
- Event-wise: finds all frequent events satisfying intra-events constraints.
- Sequence-wise: finds all frequent sequences satisfying inter-events constraints.



CAMLS

- Event-wise
  - Iterative approach of candidate-generation-and-test, based on the apriori property
  - The intra-event constraints are integrated within the process, making it more efficient



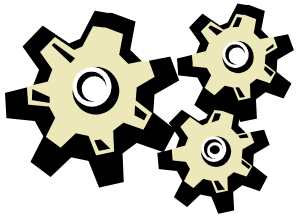
CAMLS

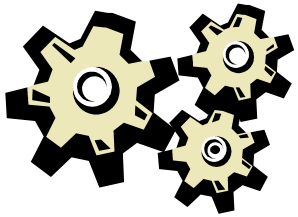
- Sequence-wise
  - Iterative approach of candidate-generation-and-test, based on the apriori property
  - The inter-events constraints are integrated within the process, making it more efficient
  - Uses the occurrence index for efficient support calculation.
  - A novel pruning strategy for redundant candidates

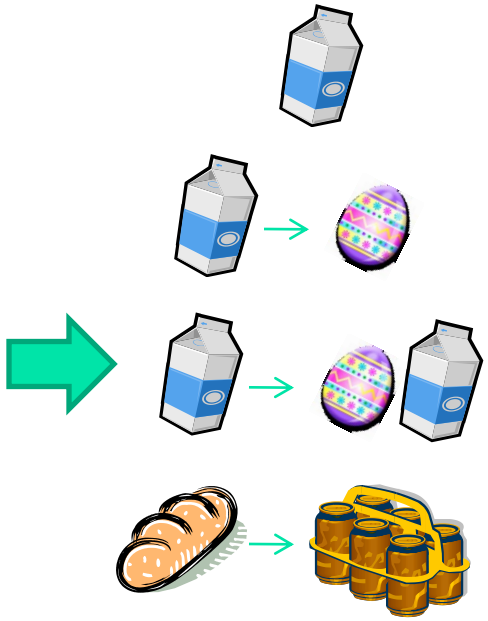
CAMLS Overview

Input **Event-wise** **Sequence-wise** **Output**

Constraints (*minSup*, *maxGap*, …)







Constrained radix-ordered frequent events + occurrence index

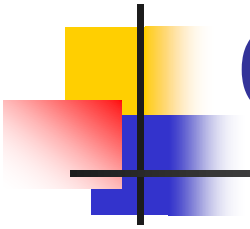
Event-wise

Example soon!

1. L_1 = all frequent **items**
2. **for** $k=2; L_{k-1} \neq \Phi; k++$ **do**
   1. generateCandidates(L_{k-1})
   2. L_k = pruneCandidates()
3. **end for**

Constraints such as Singletons or Maxitemsize are checked here

Prune, calculate support count and create occurrence index



Occurrence Index

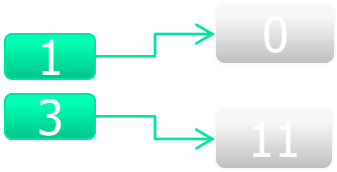
- a compact representation of all occurrences of a sequence
- Structure: list of *sids*, each associated with a list of *eids*

Event-wise Example

Input
minSup=2

sid	eid	event
1	0	*(acd)*
1	5	*(bcd)*
1	10	*b*
2	0	*a*
2	4	*c*
2	8	*(bd)*
3	0	*(cde)*
3	7	*e*
3	11	*(acd)*

All frequent items:
a:3, b:2, c:3, d:3



candidates:
(ab),(ac),(ad),
(bc),…

Support count:
(ac):2, (ad):2,
(bd):2, (cd):2

candidates:
(abc),
(abd),(acd),…

Support count:
(acd):2

No more candidates!

Sequence-wise

1. L_1 = all frequent **1-sequences**
2. **for** $k=2; L_{k-1} \neq \Phi; k++$ **do**
   1. generateCandidates(L_{k-1})
   2. L_k = pruneAndSupCalc()
3. **end for**

Sequence-wise Candidate Generation

- If two frequent *k*-sequences *s′* and *s″* share a common *k-1* prefix and s_1 is a *generator*, we form a new candidate

$s' = \langle s'_1 s'_2 \ldots s'_k \rangle$ $\quad s'' = \langle s''_1 s''_2 \ldots s''_k \rangle$

$\langle s'_1 s'_2 \ldots s'_{k-1} \rangle = \langle s''_1 s''_2 \ldots s''_{k-1} \rangle$

- Note that sequences grows not by one item but by all frequent events found in the first phase – i.e s''_k may be an event composed of a set of items

Sequence-wise Generator

- *maxGap* is a special kind of constraint in two ways:
  - Highly data dependant
  - Apriori property may not be applicable
- A frequent sequence that does not satisfy *maxGap* is flagged as **non-generator**.
- Example:
- Assume <ab> is frequent but may be pruned because the distance between a and b > maxgap
- But there are frequent sequences <ac> and <bc> and in <acb> all maxgap constraints are ok!
- So <ab> becomes a non-Generator but is kept in order not to prune <acb>…!

Sequence-wise Pruning: How its done

1. Keep a radix-ordered list of pruned sequences in current iteration
2. In the same iteration, one generate k-sequences from events of different size. Its possible that a k-sequence will contain another k-sequence in the same iteration.
3. With a new candidate:
    1. Check subsequence in pruned list
    2. Test for frequency
    3. Add to pruned list if needed

    For example: if k-sequence <abc> was found infrequent and k-sequence <a(bd)c> was generated because both b and bd are frequent, then <a(bd)c> can be pruned – this type of pruning is special to CAMLS

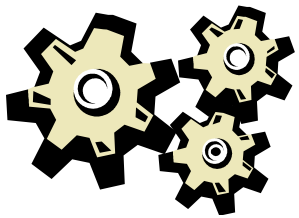
Example

Original DB

sid	eid	event
1	0	*(acd)*
1	5	*(bcd)*
1	10	*b*
2	0	*a*
2	4	*c*
2	8	*(bd)*
3	0	*(cde)*
3	7	*e*
3	11	*(acd)*

minSup=2
maxGap=5

Event-wise



<(a)> : 3	g
<(b)> : 2	g
<(c)> : 3	g
<(d)> : 3	g
<(ac)> : 2	g
<(ad)> : 2	g
<(bd)> : 2	g
<(cd)> : 2	g
<(acd)> : 2	g

Candidate generation

<aa>
<ab>
…
<a(acd)>
<ba>
<bb>
…
<(acd)(acd)>

<aa> is added to pruned list.
<a(ac)> is a super-sequence of <aa>, therefore it is pruned.
<ab> does not pass maxGap, therefore it is not a generator.

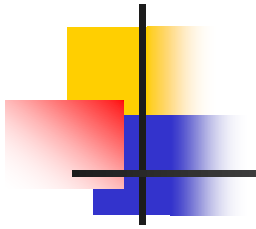
<ab> : 2	
<ac> : 2	g
<ad> : 2	
<a(bd)> : 2	
<cb> : 2	g
<cd> : 2	g
<c(bd)> : 2	g
<dc> : 2	
<dd> : 2	

<acb>
<acd>
…

<acb>:2

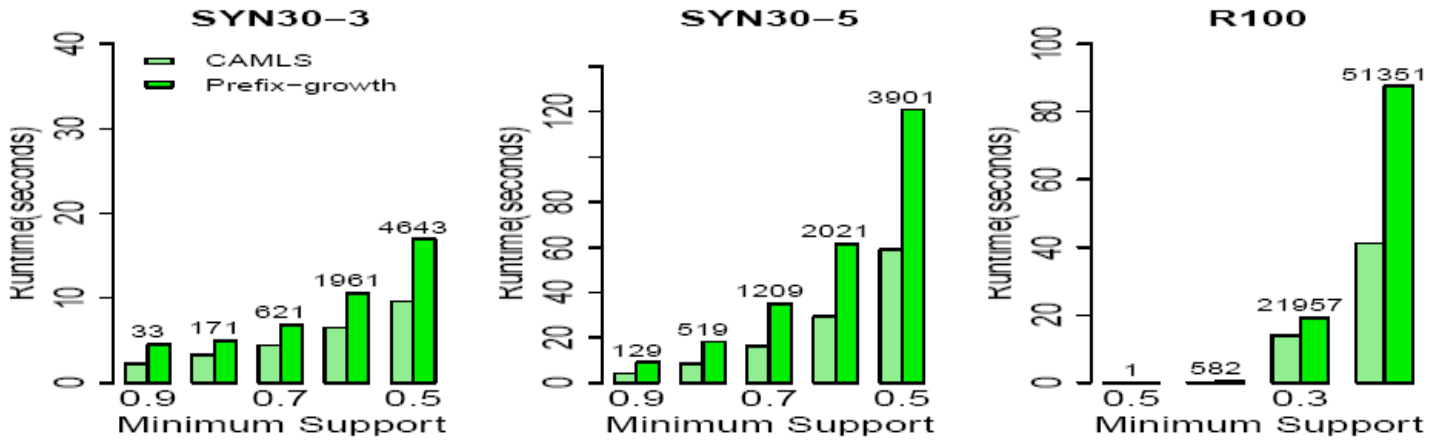
No more candidates!

Evaluation - Tested Domains

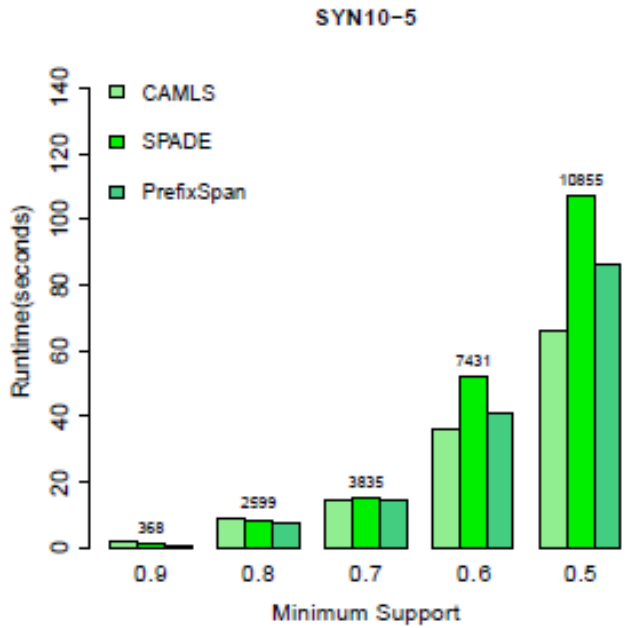


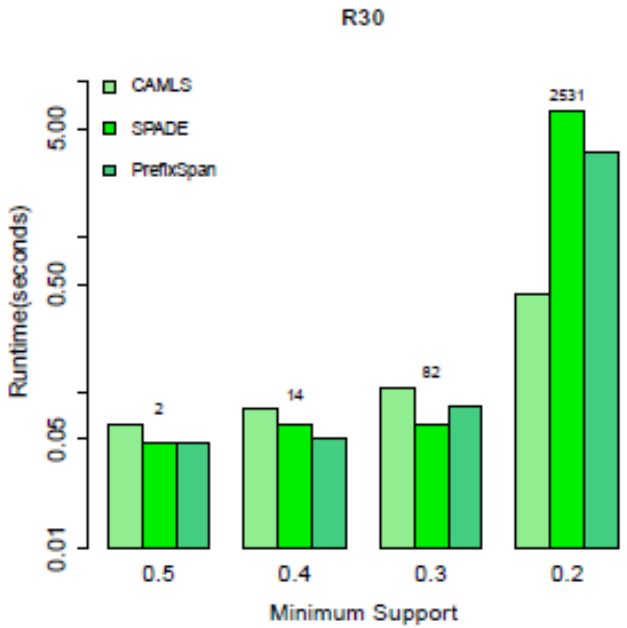
- Predicting machine failures –
  - Syn-m stands for a synthetic database simulating the machine behavior with m meta-features
- Real Stocks data values
  - Rn stands for stock data (10 different stocks) for n days
- Number above rectangles indicate number of patterns found
- Note, both domains require intelligent pre-processing and discretization

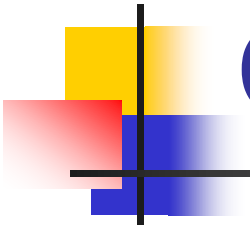
CAMLS Compared with PrefixSpan



CAMLS Compared with Spade and PrefixSpan

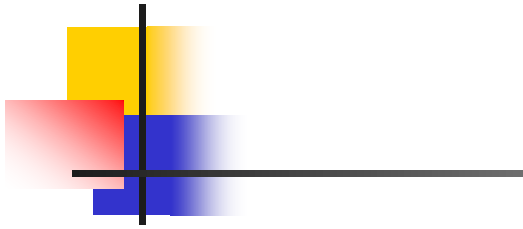






Conclusions

- CAMLS outpeforms Spade and PrefixSpan when minSupp is low, i.e. when many sequences are generated



Thank You!