Sequential Pattern Mining

Sequential Pattern Mining

- Sequential Pattern and Sequential Pattern Mining
- GSP: Apriori-Based Sequential Pattern Mining
- SPADE: Sequential Pattern Mining in Vertical Data Format
- PrefixSpan: Sequential Pattern Mining by Pattern-Growth
- CloSpan: Mining Closed Sequential Patterns

Sequential Pattern and Sequential Pattern Mining

Sequence Databases & Sequential Patterns

- Sequential pattern mining has broad applications
  - Customer shopping sequences
    - Purchase a laptop first, then a digital camera, and then a smartphone, within 6 months
  - Medical treatments, natural disasters (e.g., earthquakes), science & engineering processes, stocks and markets, ...
  - Weblog click streams, calling patterns, …
  - Software engineering: Program execution sequences, …
  - Biological sequences: DNA, protein, …
- Transaction DB, sequence DB vs. time-series DB
- Gapped vs. non-gapped sequential patterns
  - Shopping sequences, clicking streams vs. biological sequences

Sequential Pattern and Sequential Pattern Mining

- Sequential pattern mining: Given a set of sequences, find the complete set of *frequent* subsequences (i.e., satisfying the min_sup threshold)

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

- An element may contain a set of *items* (also called *events*)
- 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*

Sequential Pattern Mining Algorithms

- Algorithm requirement: Efficient, scalable, finding complete set, incorporating various kinds of user-specific constraints
- The Apriori property still holds: If a subsequence s_1 is infrequent, none of s_1's super-sequences can be frequent
- Representative algorithms
  - GSP (Generalized Sequential Patterns): Srikant & Agrawal @ EDBT'96)
  - Vertical format-based mining: SPADE (Zaki@Machine Leanining'00)
  - Pattern-growth methods: PrefixSpan (Pei, et al. @TKDE'04)
- Mining closed sequential patterns: CloSpan (Yan, et al. @SDM'03)
- Constraint-based sequential pattern mining (to be covered in the constraint mining section)

GSP: Apriori-Based Sequential Pattern Mining

GSP: Apriori-Based Sequential Pattern Mining

- Initial candidates: All 8-singleton sequences
  - <a>, <b>, <c>, <d>, <e>, <f>, <g>, <h>
- Scan DB once, count support for each candidate
- Generate length-2 candidate sequences

min_sup = 2

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

	<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>						

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

- Without Apriori pruning:

  (8 singletons) 8*8+8*7/2 = 92 length-2 candidates
- With pruning, length-2 candidates: 36 + 15= 51

GSP (Generalized Sequential Patterns): Srikant & Agrawal @ EDBT'96)

GSP Mining and Pruning

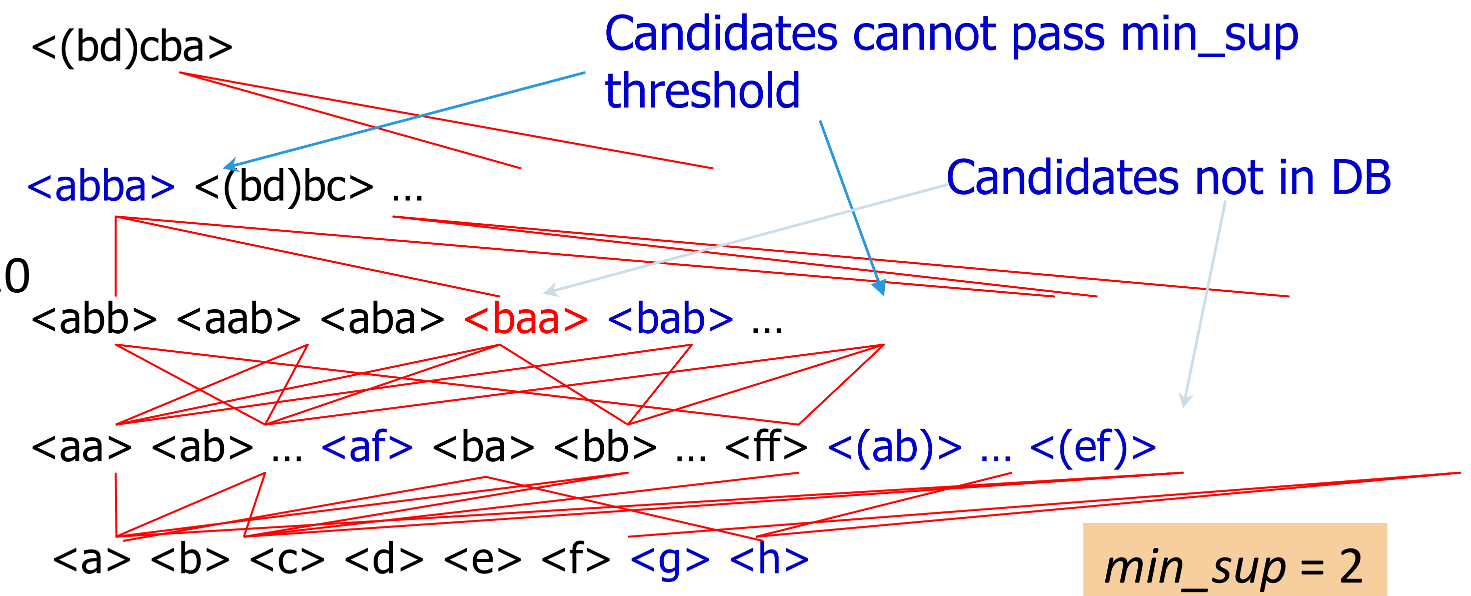
5th scan: 1 cand. 1 length-5 seq. pat.

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>
Candidates cannot pass min_sup threshold
<abba> <(bd)bc> …
Candidates not in DB
<abb> <aab> <aba> <baa> <bab> …
<aa> <ab> … <af> <ba> <bb> … <ff> <(ab)> … <(ef)>
<a> <b> <c> <d> <e> <f> <g> <h>

- Repeat (for each level (i.e., length-k))
  - Scan DB to find length-k frequent sequences
  - Generate length-(k+1) candidate sequences from length-k frequent sequences using Apriori
  - set k = k+1
- Until no frequent sequence or no candidate can be found

min_sup = 2

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

SPADE: Sequential Pattern Mining in Vertical Data Format

Sequential Pattern Mining in Vertical Data Format: The SPADE Algorithm

- A sequence database is mapped to: <SID, EID>
- Grow the subsequences (patterns) one item at a time by Apriori candidate generation

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

min_sup = 2

Ref: SPADE (Sequential PAttern Discovery using Equivalent Class) [M. Zaki 2001]

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	

PrefixSpan: Sequential Pattern Mining by Pattern-Growth

PrefixSpan: A Pattern-Growth Approach

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

min_sup = 2

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

- Prefix and suffix
  - Given <a(abc)(ac)d(cf)>
  - Prefixes: <a>, <aa>, <a(ab)>, <a(abc)>, …
  - Suffix: Prefixes-based projection

- PrefixSpan Mining: Prefix Projections
  - Step 1: Find length-1 sequential patterns
    - <a>, <b>, <c>, <d>, <e>, <f>
  - Step 2: Divide search space and mine each projected DB
    - <a>-projected DB,
    - <b>-projected DB,
    - …
    - <f>-projected DB, …

PrefixSpan (Prefix-projected Sequential pattern mining) Pei, et al. @TKDE'04

PrefixSpan: Mining Prefix-Projected DBs

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

min_sup = 2

Length-1 sequential patterns
<a>, <b>, <c>, <d>, <e>, <f>

prefix <a>

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

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

prefix <b>

<b>-projected DB

… …

prefix <c>, …, <f>

…

prefix <aa>

<aa>-projected DB

..

prefix <af>

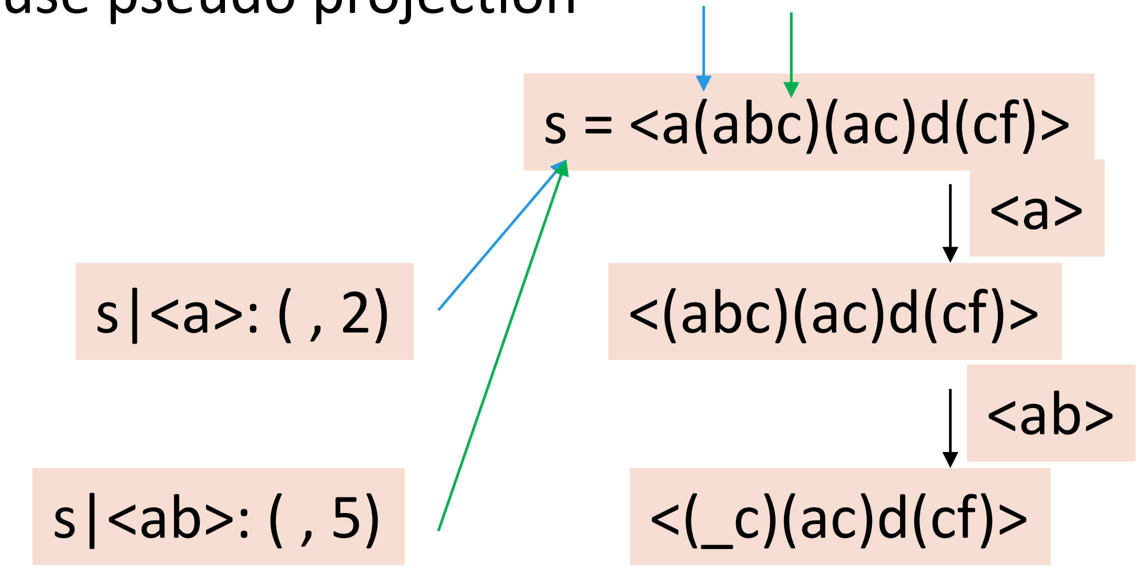
<af>-projected DB

Major strength of PrefixSpan:

- No candidate subseqs. to be generated
- Projected DBs keep shrinking

Implementation Consideration: Pseudo-Projection vs. Physical Projection

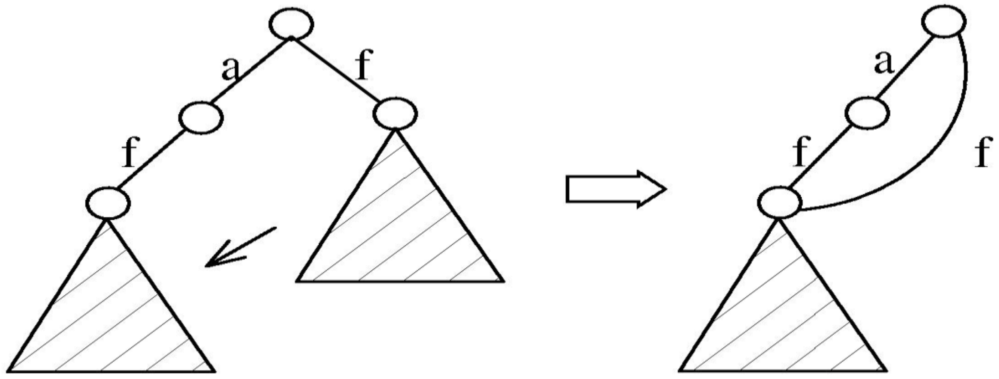
- Major cost of PrefixSpan: Constructing projected DBs
    - Suffixes largely repeating in recursive projected DBs
- When DB can be held in main memory, use pseudo projection
    - No physically copying suffixes
    - Pointer to the sequence
    - Offset of the suffix
- But if it does not fit in memory
    - Physical projection
- Suggested approach:
    - Integration of physical and pseudo-projection
    - Swapping to pseudo-projection when the data fits in memory

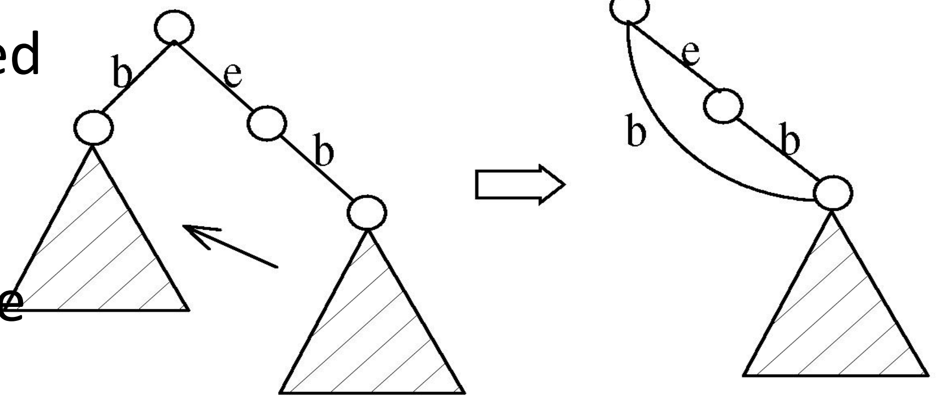


CloSpan: Mining Closed Sequential Patterns

CloSpan: Mining Closed Sequential Patterns

- A closed sequential pattern *s*: There exists no superpattern *s′* such that *s′* ɔ *s*, and *s′* and *s* have the same support
- Which ones are closed? <abc>: 20, <abcd>:20, <abcde>: 15
- Why directly mine closed sequential patterns?
  - Reduce # of (redundant) patterns
  - Attain the same expressive power
- Property P: Given two sequences s and s′, if s is a subsequence of s′, then the projected database of s = the projected database of s′ iff the size of the two projected databases are the same.
- Explore *Backward Subpattern* and *Backward Superpattern* pruning to prune redundant search space
- Greatly enhances efficiency (Yan, et al., SDM’03)



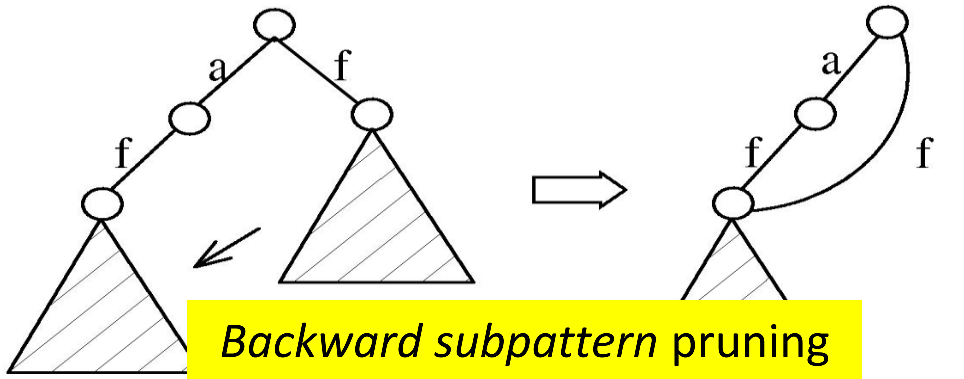


CloSpan: When Two Projected DBs Have the Same Size

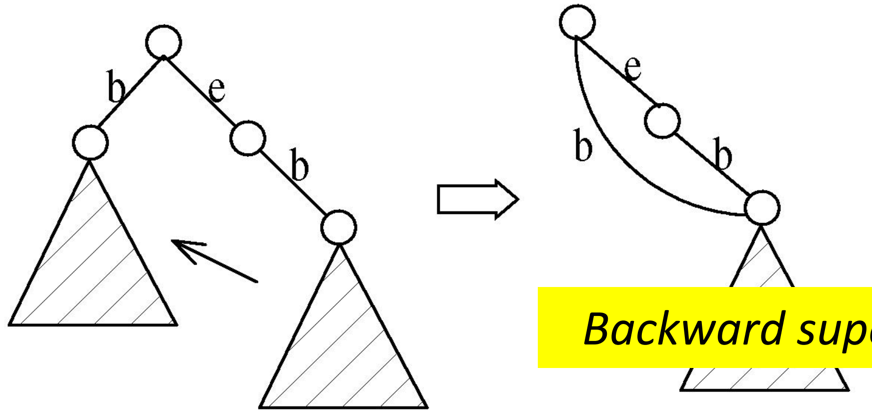
- Exploring Property P for closed pattern mining
- When two projected sequence DBs have the same size?
  - Here is one example:

ID	Sequence
1	<aefbcg>
2	<afegb(ac)>
3	<afea>

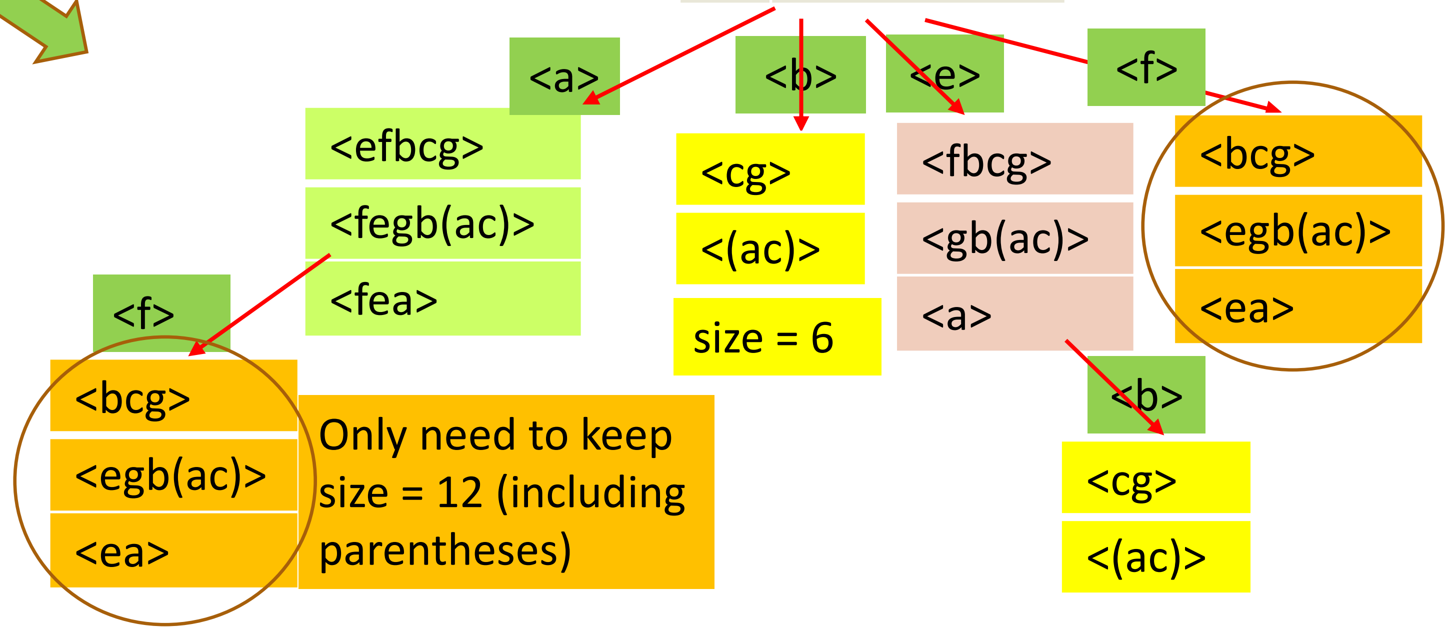
min_sup = 2



Backward subpattern pruning



Backward superpattern pruning



Only need to keep size = 12 (including parentheses)

Summary

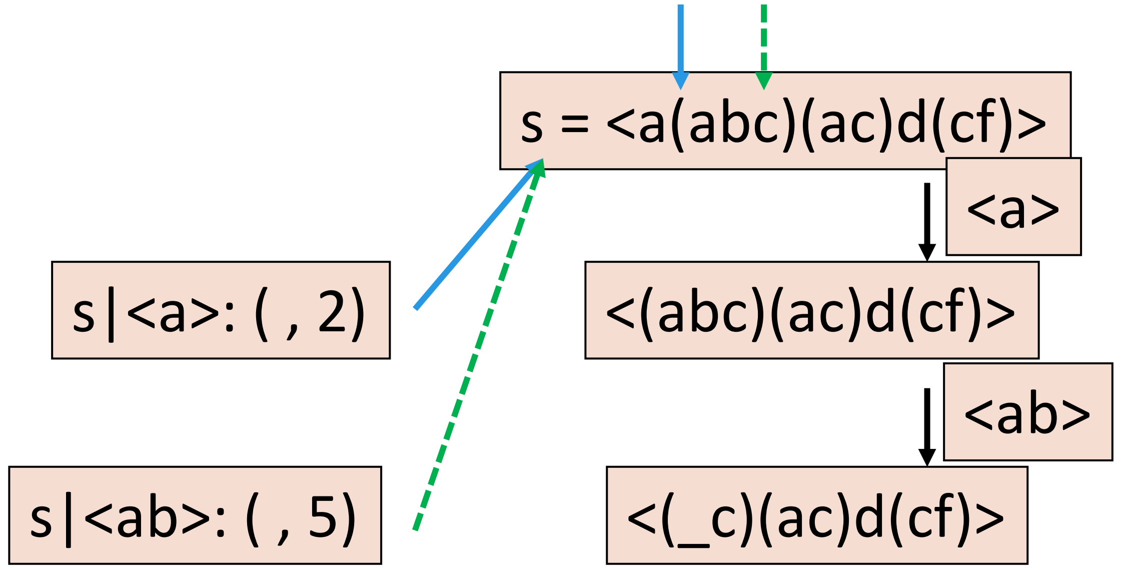
Summary: Sequential Pattern Mining

- Concepts of Sequential Pattern Mining
- Sequential Pattern Mining Algorithms
  - GSP (Generalized Sequential Patterns)
  - Vertical Format-Based Mining: SPADE
  - Pattern-Growth Methods: PrefixSpan
- Mining Closed Sequential Patterns: CloSpan

Recommended Readings

- R. Srikant and R. Agrawal, “Mining sequential patterns: Generalizations and performance improvements”, EDBT’96
- M. Zaki, “SPADE: An Efficient Algorithm for Mining Frequent Sequences”, Machine Learning, 2001
- J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu, "Mining Sequential Patterns by Pattern-Growth: The PrefixSpan Approach", IEEE TKDE, 16(10), 2004
- X. Yan, J. Han, and R. Afshar, “CloSpan: Mining Closed Sequential Patterns in Large Datasets”, SDM'03

Pseudo-Projection vs. Physical Projection



Graph Pattern Mining

Graph Pattern Mining

- ❑ Graph Pattern and Graph Pattern Mining
- ❑ Apriori-Based Graph Pattern Mining Methods
- ❑ gSpan: A Pattern-Growth-Based Method
- ❑ CloseGraph: Mining Closed Graph Patterns
- ❑ Graph Pattern Mining Application I: Graph Indexing
- ❑ Graph Pattern Mining Application II: Graph Similarity Search

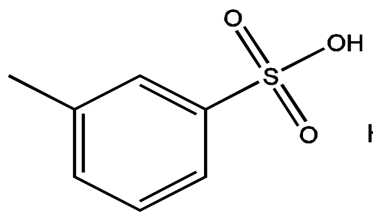
Thanks to Xifeng Yan@UCSB and Feida Zhu@SMU.SG for their contributions

Graph Pattern and Graph Pattern Mining

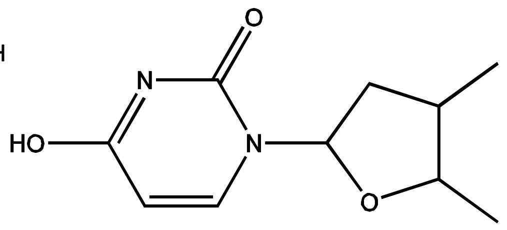
Frequent (Sub)Graph Patterns

- Given a labeled graph dataset D = $\{G_1, G_2, \ldots, G_n)$, the supporting graph set of a subgraph g is $D_g = \{G_i \mid g \subseteq G_i, G_i \in D\}$
  - $\text{support}(g) = |D_g| / |D|$
- A (sub)graph g is ***frequent*** if *support*(g) ≥ *min_sup*
- Ex.: Chemical structures
- Alternative:
  - Mining frequent subgraph patterns from a single large graph or network

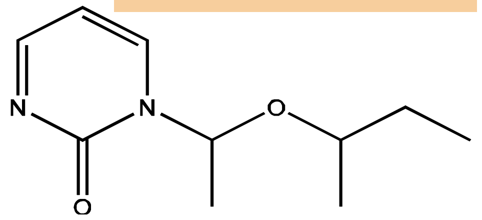
Graph Dataset



(A)



(B)

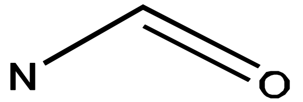


(C)

min_sup = 2

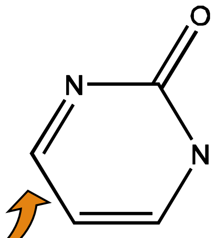
Frequent Graph Patterns

(1)



support = 67%

(2)



Applications of Graph Pattern Mining

- Bioinformatics
  - Gene networks, protein interactions, metabolic pathways
- Chem-informatics: Mining chemical compound structures
- Social networks, web communities, tweets, …
- Cell phone networks, computer networks, …
- Web graphs, XML structures, Semantic Web, information networks
- Software engineering: Program execution flow analysis
- Building blocks for graph classification, clustering, compression, comparison, and correlation analysis
- Graph indexing and graph similarity search

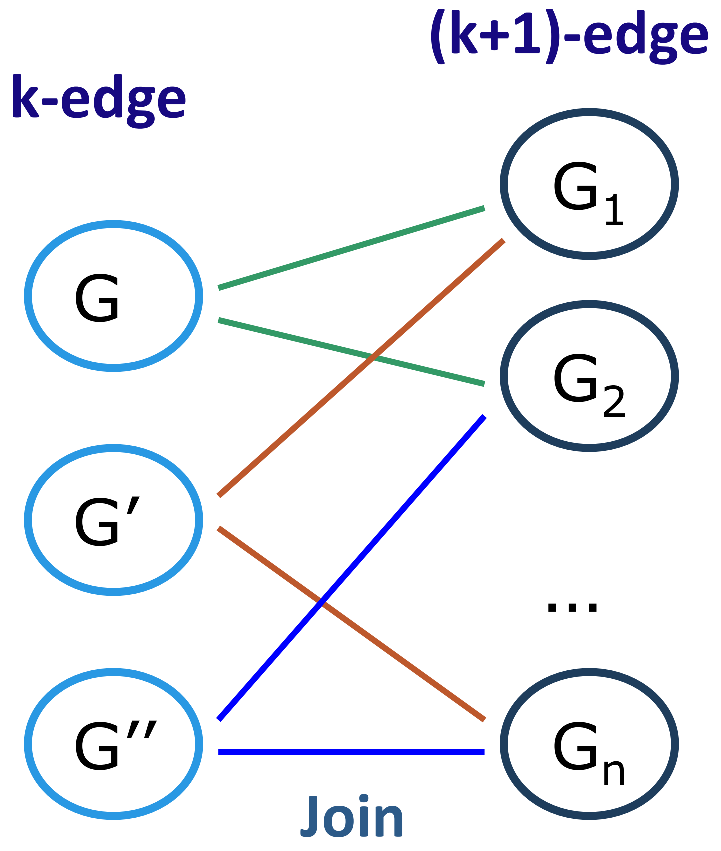
Graph Pattern Mining Algorithms: Different Methodologies

- Generation of candidate subgraphs
  - Apriori vs. pattern growth (e.g., FSG vs. gSpan)
- Search order
  - Breadth vs. depth
- Elimination of duplicate subgraphs
  - Passive vs. active (e.g., gSpan [Yan & Han, 2002])
- Support calculation
  - Store embeddings (e.g., GASTON [Nijssen & Kok, 2004], FFSM [Huan, Wang, & Prins, 2003], MoFa [Borgelt & Berthold, ICDM'02])
- Order of pattern discovery
  - Path → tree → graph (e.g., GASTON [Nijssen & Kok, 2004])

Graph Pattern Mining: Apriori-Based Approach

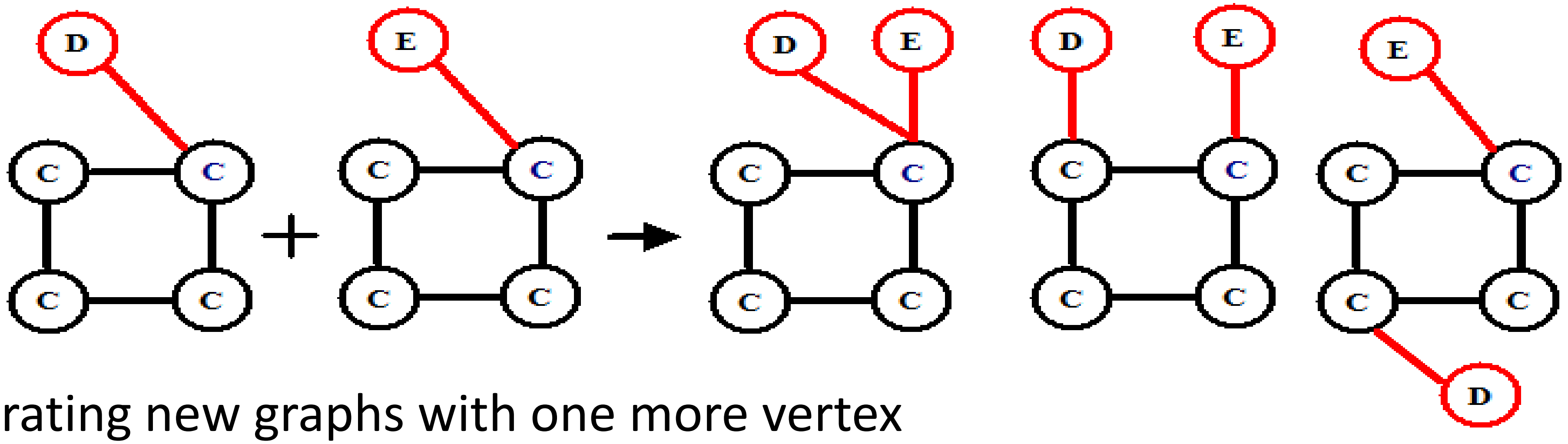
Apriori-Based Approach

- The Apriori property (anti-monotonicity): A size-*k* subgraph is frequent only if all of its subgraphs are frequent
- A candidate size-(*k*+1) edge/vertex subgraph is generated if its corresponding two *k*-edge/vertex subgraphs are frequent
- Iterative mining process:
  - Candidate-generation → candidate pruning → support counting → candidate elimination



Candidate Generation: Vertex Growing vs. Edge Growing

- Methodology: Breadth-search, Apriori joining two size-*k* graphs
  - Many possibilities at generating size-(*k*+1) candidate graphs

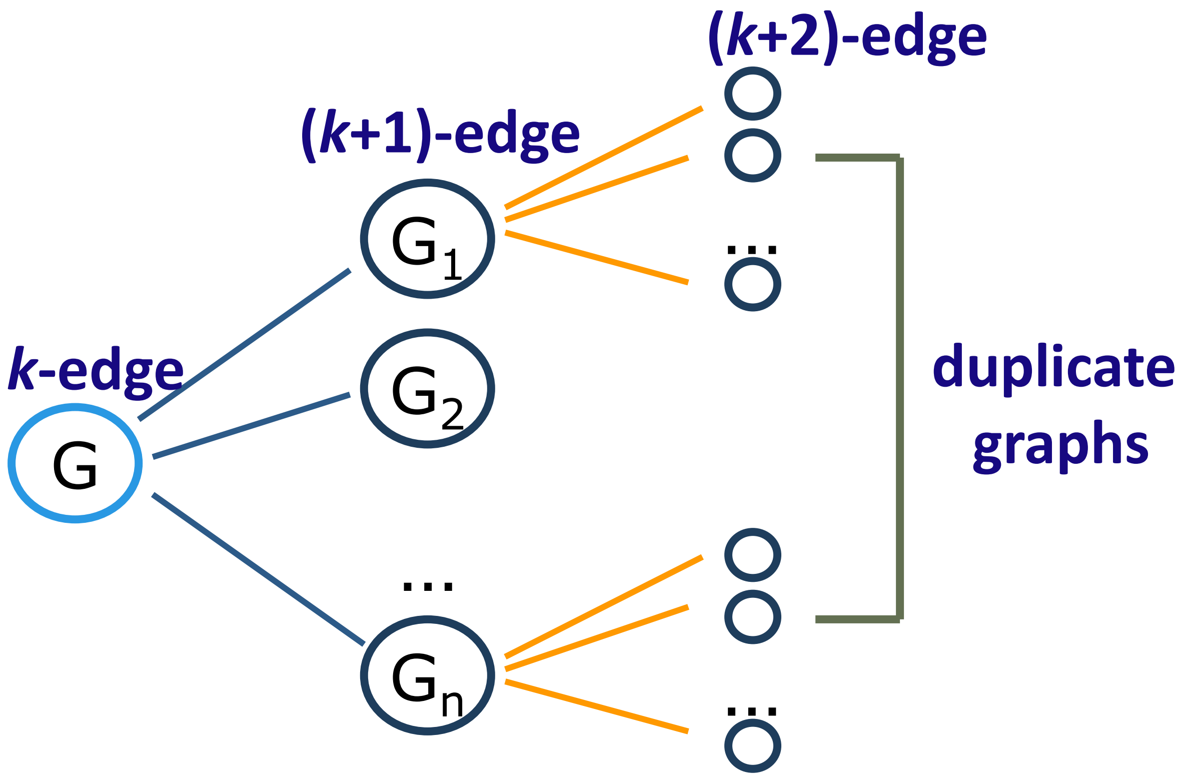


- Generating new graphs with one more vertex
  - AGM (Inokuchi, Washio, & Motoda, PKDD'00)
- Generating new graphs with one more edge
  - FSG (Kuramochi & Karypis, ICDM'01)
- Performance shows *via edge growing* is more efficient

gSpan: A Pattern Growth Approach

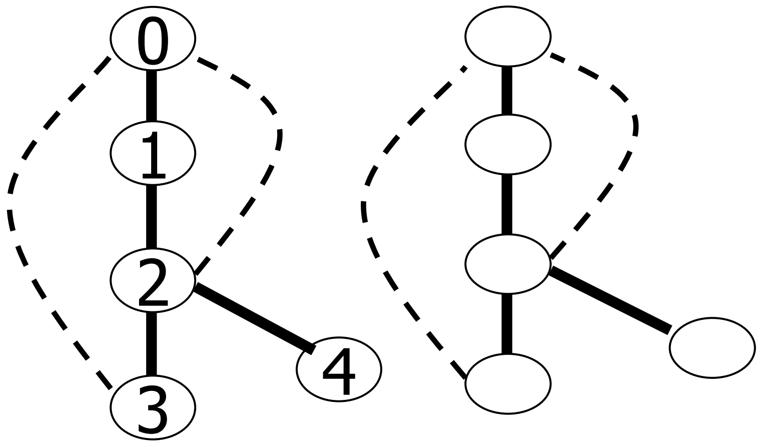
Pattern-Growth Approach

- Depth-first growth of subgraphs from *k*-edge to (*k*+1)-edge, then (*k*+2)-edge subgraphs
- Major challenge
  - Generating many duplicate subgraphs
- Major idea to solve the problem
  - Define an order to generate subgraphs
  - DFS spanning tree: Flatten a graph into a sequence using depth-first search
  - gSpan (Yan & Han, ICDM'02)



gSPAN: Graph Pattern Growth in Order

- **Right-most path extension** in subgraph pattern growth
  - Right-most path: The path from root to the right-most leaf (choose the vertex with the smallest index at each step)
  - Reduce generation of duplicate subgraphs
- **Completeness:** The enumeration of graphs using right-most path extension is <u>complete</u>
- DFS code: Flatten a graph into a sequence using depth-first search

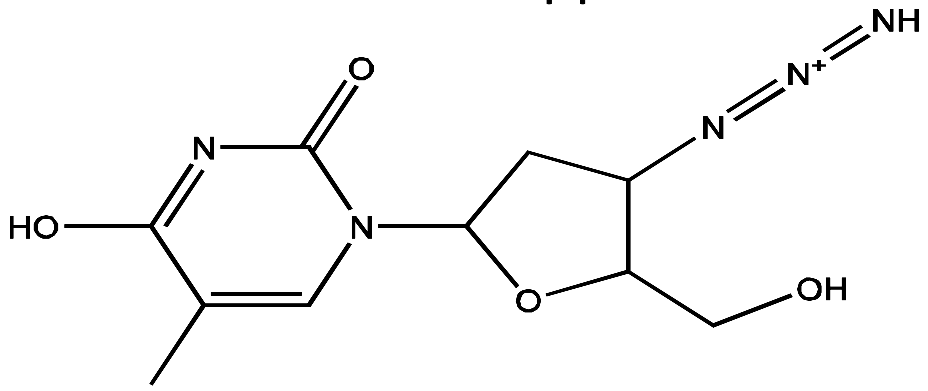


e_0: (0,1)
e_1: (1,2)
e_2: (2,0)
e_3: (2,3)
e_4: (3,0)
e_5: (2,4)

CloseGraph: Mining Closed Graph Patterns

Why Mine Closed Graph Patterns?

- Challenge: An **n**-edge frequent graph may have 2^n subgraphs
- Motivation: Explore *closed frequent subgraphs* to handle graph pattern explosion problem
- A frequent graph G is *closed* if there exists no supergraph of G that carries the same support as G

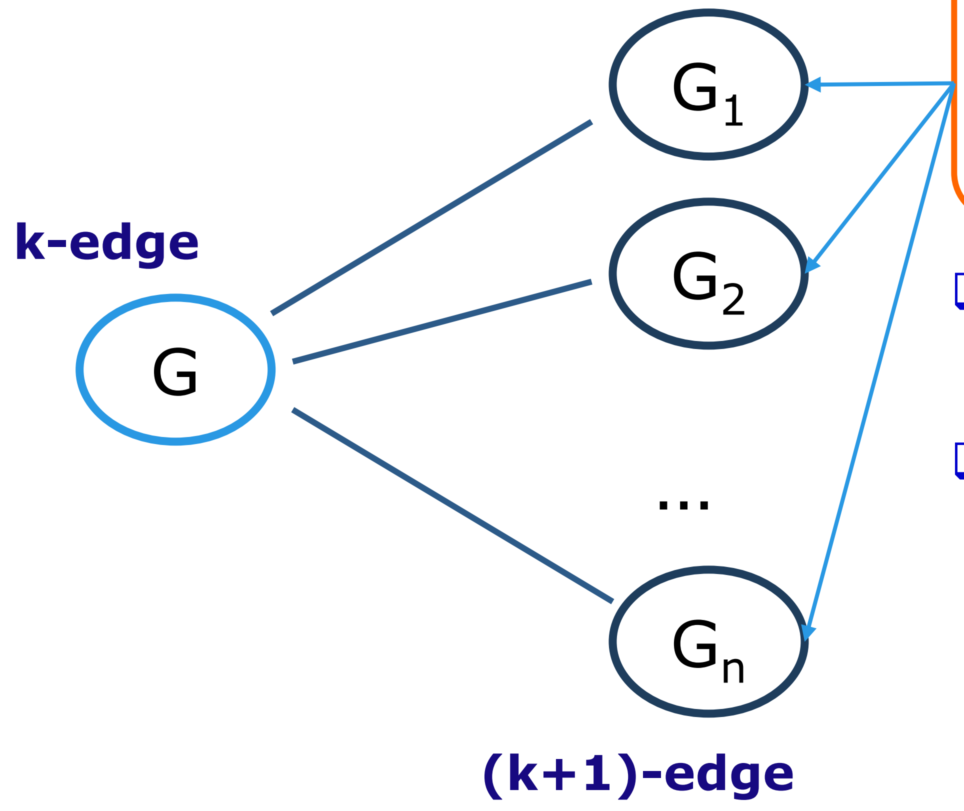


If this subgraph is *closed* in the graph dataset, it implies that none of its frequent super-graphs carries the same support

- *Lossless compression:* Does not contain non-closed graphs, but still ensures that the mining result is complete
- Algorithm CloseGraph: Mines closed graph patterns directly

CloseGraph: Directly Mining Closed Graph Patterns

- CloseGraph: Mining closed graph patterns by extending gSpan (Yan & Han, KDD'03)



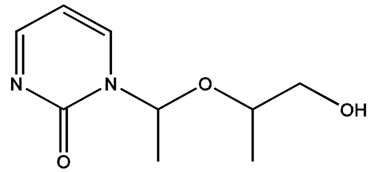
At what condition can we stop searching their children, i.e., **early termination?**

- Suppose G and G_1 are frequent, and G is a subgraph of G_1
- If **in any part of the graph in the dataset where G occurs, G_1 also occurs**, then we need not grow G (except some special, subtle cases), since none of G's children will be closed except those of G_1

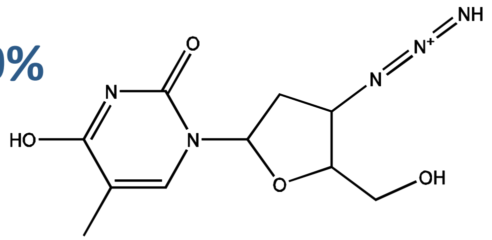
Experiment and Performance Comparison

- The AIDS antiviral screen compound dataset from NCI/NIH
- The dataset contains 43,905 chemical compounds
- Discovered patterns: The smaller minimum support, the bigger and more interesting subgraph patterns discovered

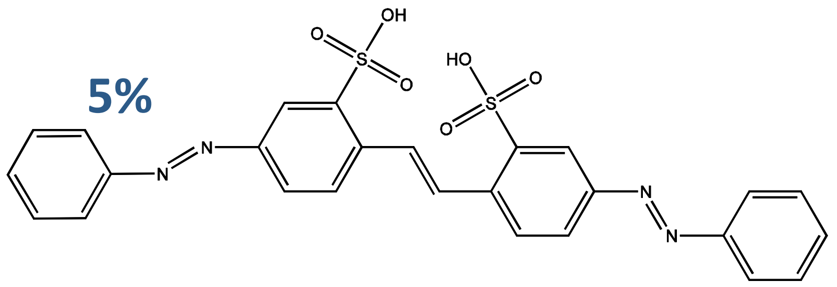
20%



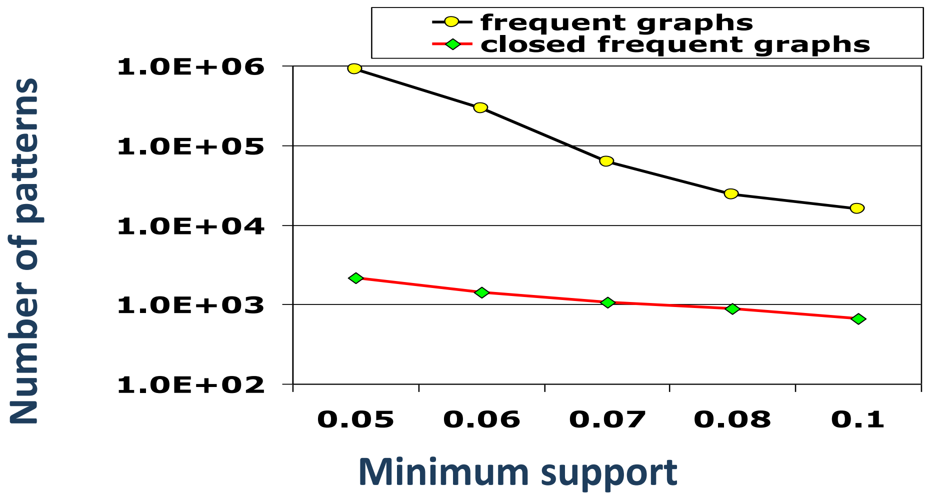
10%



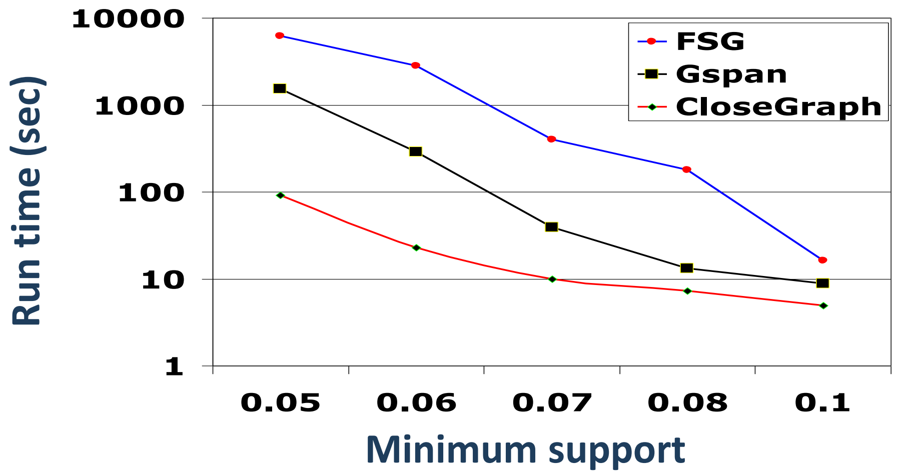
5%



# of Patterns: Frequent vs. Closed



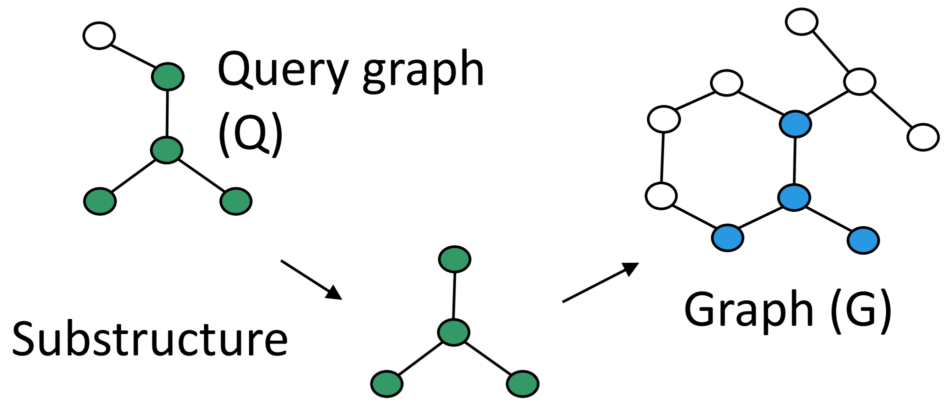
Runtime: Frequent vs. Closed



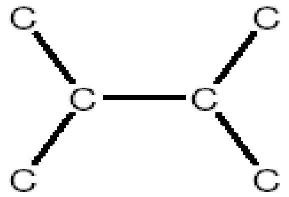
Graph Pattern Mining Application I: Graph Indexing

Application of Pattern Mining I: Graph Indexing

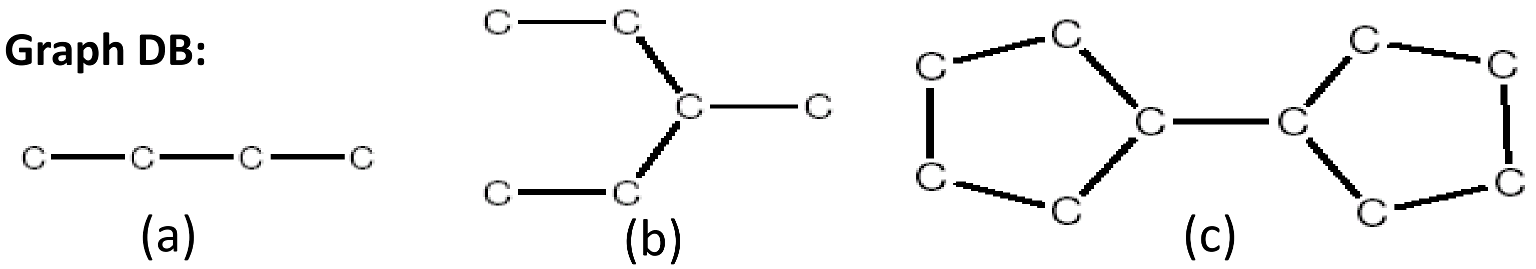
- Graph query: Find all the graphs in a graph DB containing a given query graph
- Index should be a powerful tool
- Path-index may not work well
- Solution: Index directly on substructures (i.e., graphs)



Query Q:



Graph DB:

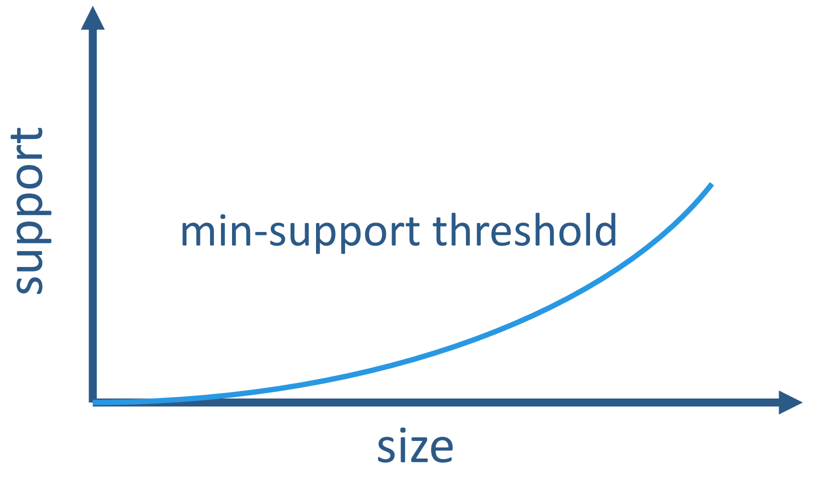


Only graph (c) contains Q

Path-indices: C, C-C, C-C-C, C-C-C-C cannot prune (a) & (b)

gIndex: Indexing Frequent and Discriminative Substructures

- Why index frequent substructures?
  - Too many substructures to index
  - Size-increasing support threshold
  - Large structures will likely be indexed well by their substructures



- Why discriminative substructures?
  - Reduce the index size by an order of magnitude
- Selection: Given a set of selected structures $f_1, f_2, .. f_n$, and a new structure *x*, the extra indexing power is measured by

$$\Pr\left(x \middle| f_1, f_2, \ldots f_n\right), f_i \subset x$$

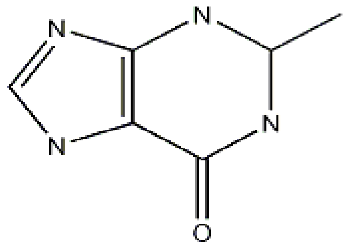
when $\Pr(x|f_1, f_2, \ldots, f_n)$ is small enough, *x* is a discriminative structure and should be included in the index

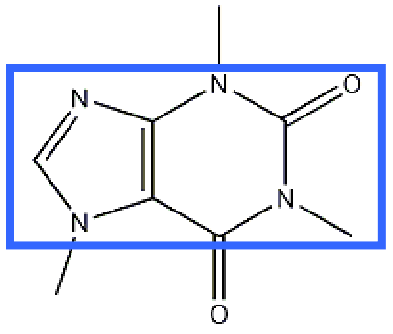
- Experiments show that gIndex is small, effective, and stable

Graph Pattern Mining Application II: Graph Similarity Search

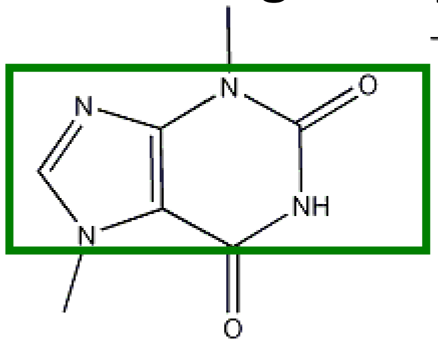
Application II: Support Substructure Similarity Search

- Find graphs in a graph DB containing substructures similar to a given query graph
- Ex. Data: A chemical compound DB
  - A query graph q:

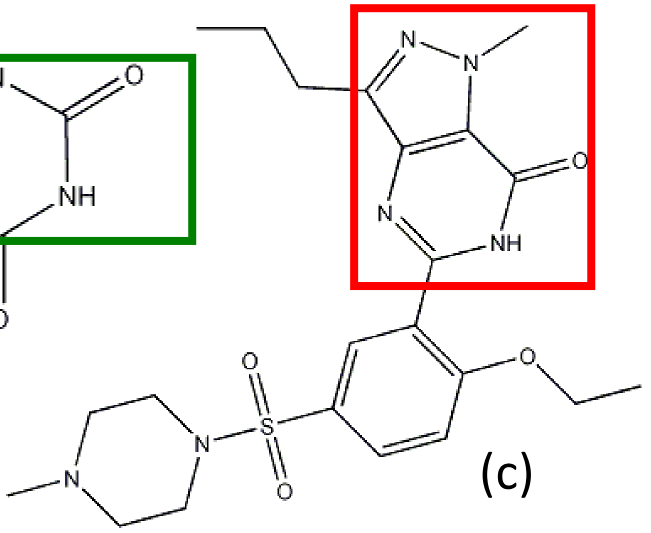




(a)



(b)

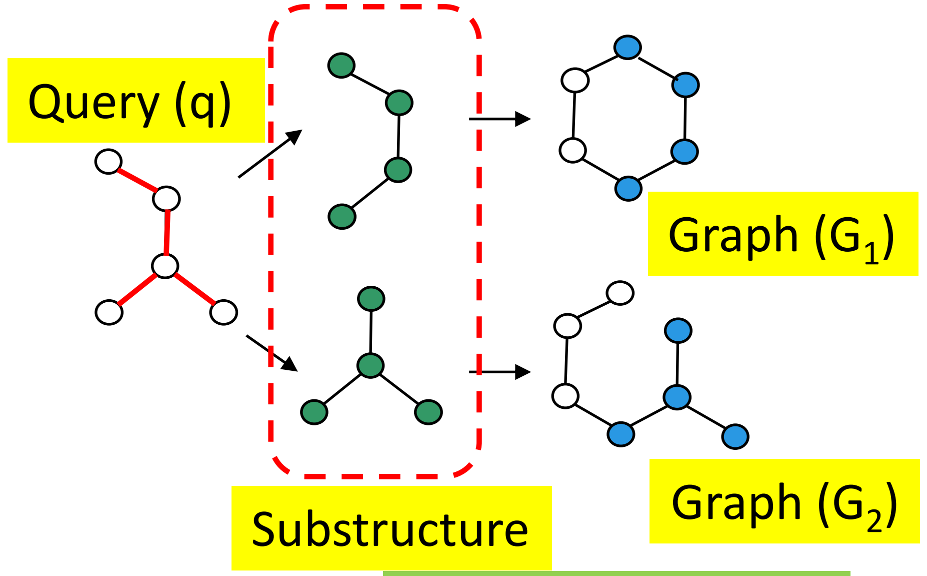


(c)

- How to do similarity search efficiently?
  - No indexing? – Sequential scan + computing subgraph similarity – too costly!
  - Build graph indices to support approximate search?
    - Need an explosive number of subgraphs to cover all the *similar* subgraphs!
- An elegant solution (Yan, Yu, & Han, SIGMOD'05):
  - Keep the graph index structure, but select features in the query space

Feature-Based Similarity Search

- Decompose a query graph into a set of features
- Feature-based similarity measure
  - Each graph is represented as a feature vector $X = \{x_1, x_2, \ldots, x_n\}$
  - Similarity is defined by the distance of their corresponding vectors
- If graph G contains the major part of a query graph q, G should share a number of common features with q
  - Given a relaxation ratio, one can calculate the maximal number of features that can be missed!



Assume: Query graph has 5 features

Relaxation threshold: Can miss at most 2 features

Then: G_1, G_2, G_3 are pruned

Graphs in database

features	G_1	G_2	G_3	G_4	G_5
f_1	0	1	0	1	1
f_2	0	1	0	0	1
f_3	1	0	1	1	1
f_4	1	0	0	0	1
f_5	0	0	1	1	0

A feature-graph matrix

Summary

Summary: Graph Pattern Mining

- Graph Pattern and Graph Pattern Mining
- Apriori-Based Graph Pattern Mining Methods
- gSpan: A Pattern-Growth-Based Method
- CloseGraph: Mining Closed Graph Patterns
- Graph Pattern Mining Application I: Graph Indexing
- Graph Pattern Mining Application II: Graph Similarity Search

Recommended Readings

- Kuramochi, M., & Karypis, G. (2001). Frequent subgraph discovery. *ICDM'01.*
- Yan, X. & Han, J. (2002). gSpan: Graph-based substructure pattern mining. *ICDM'02.*
- Yan, X., & Han, J. (2003). CloseGraph: Mining closed frequent graph patterns. *KDD'03.*
- Yan, X., Yu, P. S., & Han, J. (2004). Graph indexing: A frequent structure-based approach. *SIGMOD'04.*
- Yan, X., Yu, P. S., & Han, J. (2005). Substructure similarity search in graph databases. *SIGMOD'05.*

Additional References

- ❑ Borgelt, C. & Berthold, M. R. (2002). Mining molecular fragments: Finding relevant substructures of molecules. *ICDM'02.*
- ❑ Huan, J., Wang, W., & Prins, J. (2003). Efficient mining of frequent subgraph in the presence of isomorphism. *ICDM'03.*
- ❑ Inokuchi, A., Washio, T., & Motoda, H. (2000). An apriori-based algorithm for mining frequent substructures from graph data. *PKDD'00.*
- ❑ Nijssen, S., & Kok, J. (2004). A quickstart in frequent structure mining can make a difference. *KDD'04*
- ❑ Vanetik, N., Gudes, E., & Shimony, S. E. (2002). Computing frequent graph patterns from semistructured data*. ICDM'02.*

Pattern Discovery for Software Bug Mining

Pattern Mining Application: Software Bug Detection

- **Mining rules from source code**
  - Bugs as deviant behavior (e.g., by statistical analysis)
  - Mining programming rules (e.g., by frequent itemset mining)
  - Mining function precedence protocols (e.g., by frequent subsequence mining)
  - Revealing neglected conditions (e.g., by frequent itemset/subgraph mining)
- **Mining rules from revision histories**
  - By frequent itemset mining
- **Mining copy-paste patterns from source code**
  - Find copy-paste bugs (e.g., CP-Miner [Li et al., OSDI'04]) (to be discussed here)
    - Reference: Z. Li, S. Lu, S. Myagmar, Y. Zhou, "CP-Miner: A Tool for Finding Copy-paste and Related Bugs in Operating System Code", OSDI'04

Application Example: Mining Copy-and-Paste Bugs

- Copy-pasting is common
  - 12% in Linux file system
  - 19% in X Window system
- Copy-pasted code is error-prone
- Mine *"forget-to-change"* bugs by sequential pattern mining
  - Build a sequence database from source code
  - Mining sequential patterns
  - Finding mismatched identifier names & bugs

```
void __init prom_meminit(void)
{
    ......
    for (i=0; i<n; i++) {
        total[i].adr = list[i].addr;
        total[i].bytes = list[i].size;
        total[i].more = &total[i+1];
    }
    ......
```

```
for (i=0; i<n; i++) {
        taken[i].adr = list[i].addr;
        taken[i].bytes = list[i].size;
        taken[i].more = &total[i+1];
    }
```

Code copy-and-pasted but **forget to change** "id"!

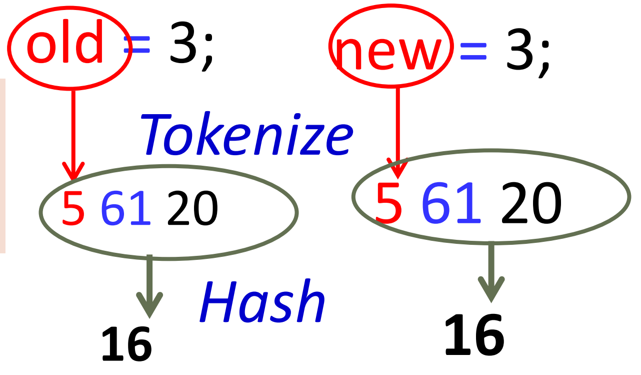
(Simplified example from *linux-2.6.6/arch/sparc/prom/memory.c*)

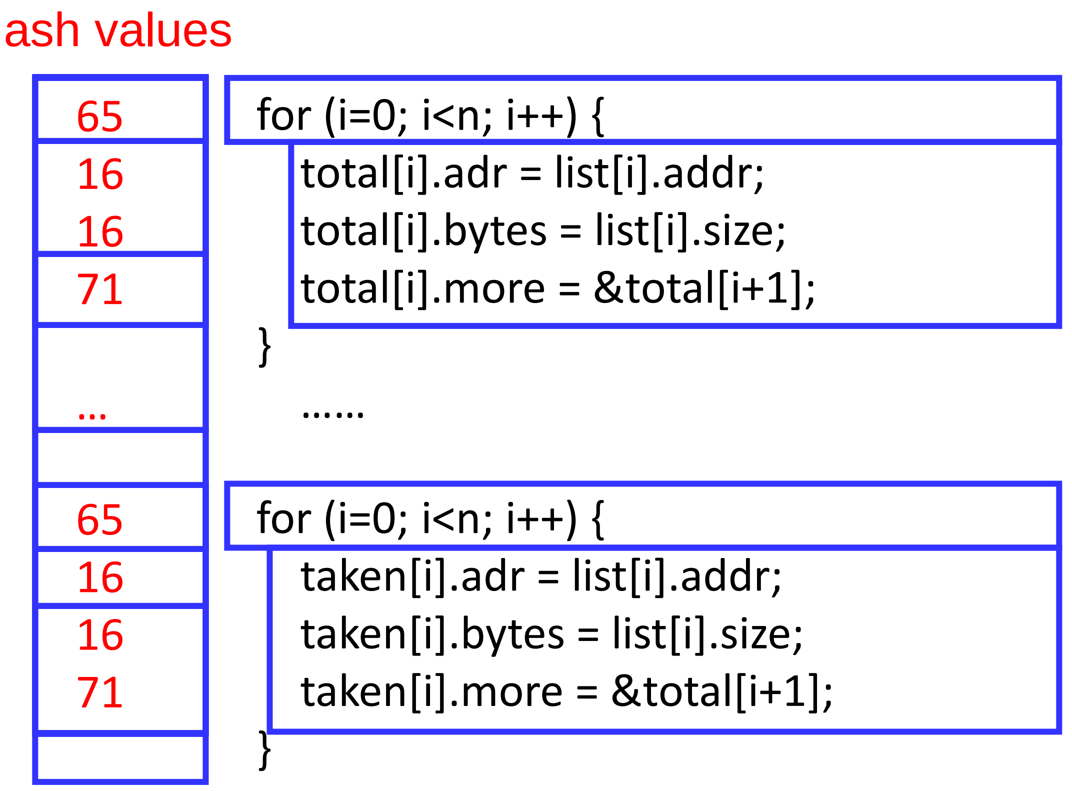
Courtesy of Yuanyuan Zhou@UCSD

Building Sequence Database from Source Code

- Statement $\xrightarrow{\text{(mapped to)}}$ number
- Tokenize each component
  - Different operators, constants, key words → different tokens
  - Same type of identifiers → same token
- Program → A long sequence
  - Cut the long sequence by blocks

Map a statement to a number





```
for (i=0; i<n; i++) {
    total[i].adr = list[i].addr;
    total[i].bytes = list[i].size;
    total[i].more = &total[i+1];
}
    ……
for (i=0; i<n; i++) {
    taken[i].adr = list[i].addr;
    taken[i].bytes = list[i].size;
    taken[i].more = &total[i+1];
}
```

Final sequence DB:
(65)
(16, 16, 71)
…
(65)
(16, 16, 71)

Courtesy of Yuanyuan Zhou@UCSD

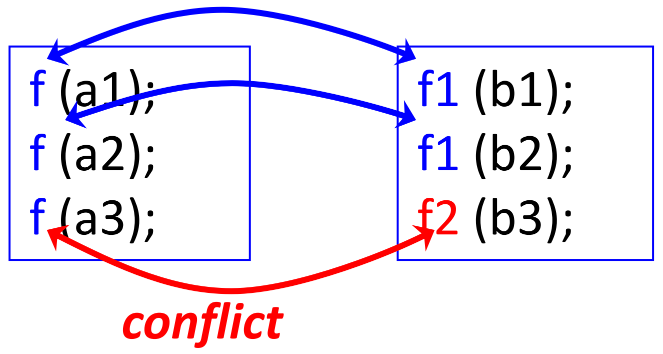
Sequential Pattern Mining & Detecting "Forget-to-Change" Bugs

- Modification to the *sequence pattern mining algorithm*
  - Constrain the max gap

(16, 16, 71)
……
(16, 16, 10, 71)

Allow a maximal gap: inserting statements in copy-and-paste

- Composing Larger Copy-Pasted Segments
  - Combine the neighboring copy-pasted segments repeatedly
- Find conflicts: Identify names that cannot be mapped to the corresponding ones
  - E.g., 1 out of 4 "**total**" is unchanged, *unchanged ratio* = 0.25
  - If 0 < *unchanged ratio* < *threshold*, then report it as a bug
- CP-Miner reported many C-P bugs in Linux, Apache, … out of millions of LOC (lines of code)



Courtesy of Yuanyuan Zhou@UCSD