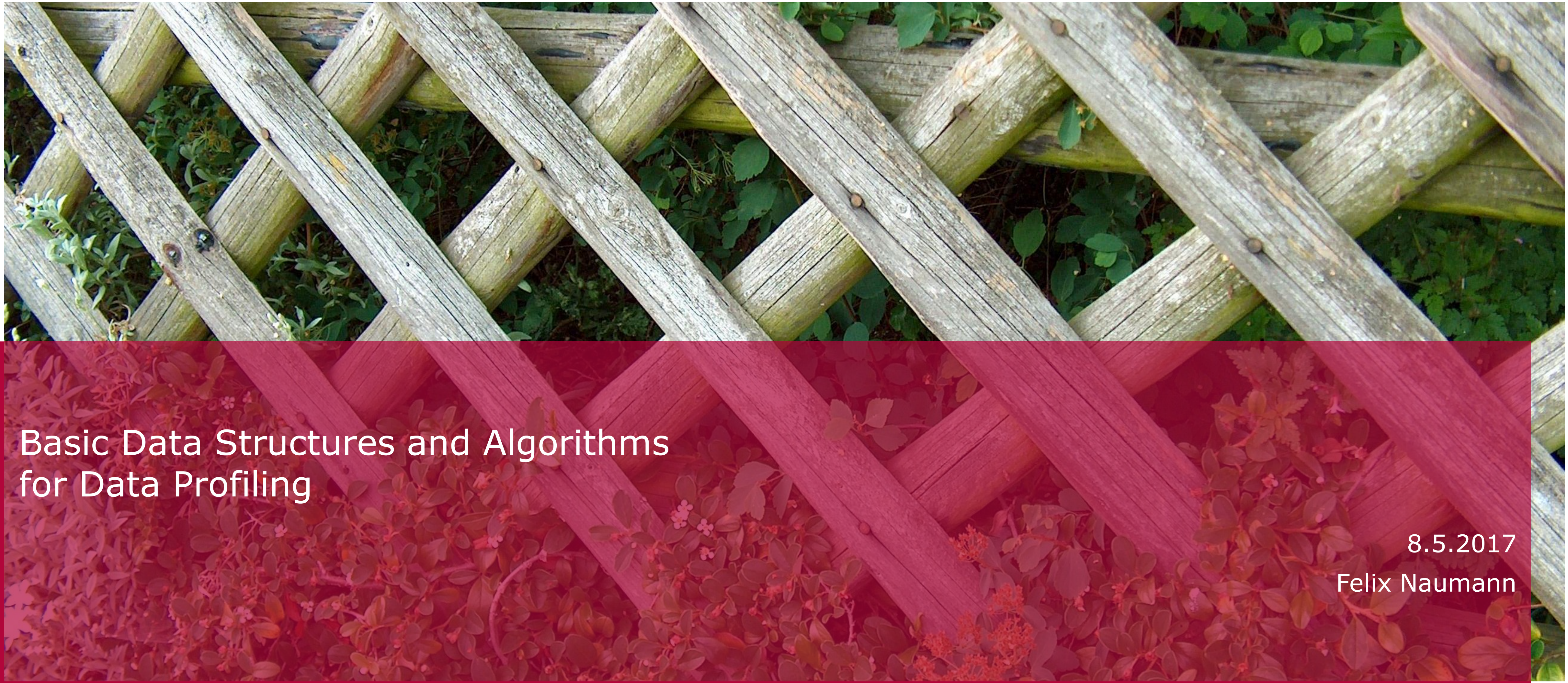
Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam



Basic Data Structures and Algorithms for Data Profiling

8.5.2017

Felix Naumann

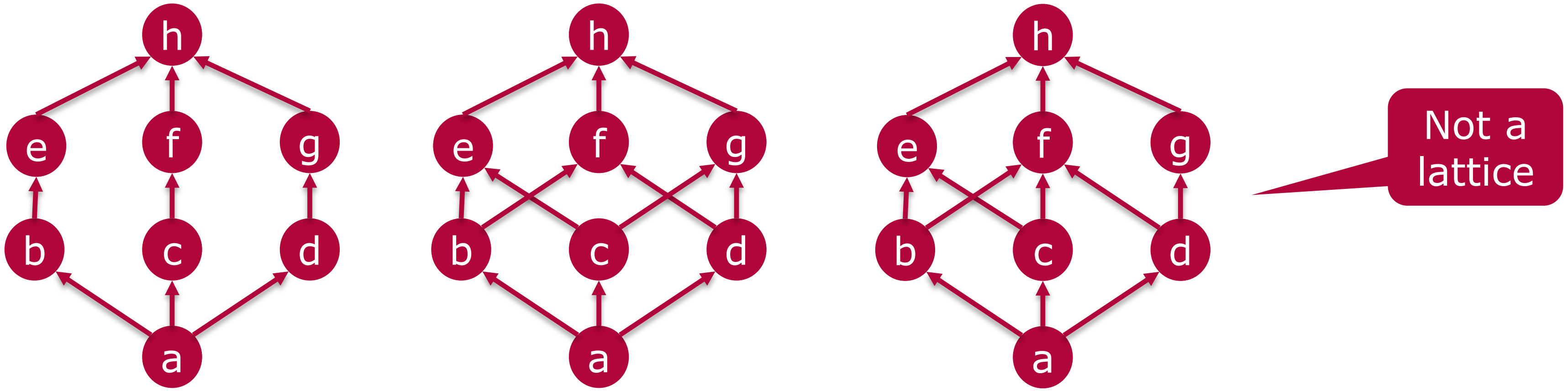
Overview

1. **The lattice**
2. Apriori lattice traversal
3. Position List Indices
4. Bloom filters

Slides with Thorsten Papenbrock

Definitions

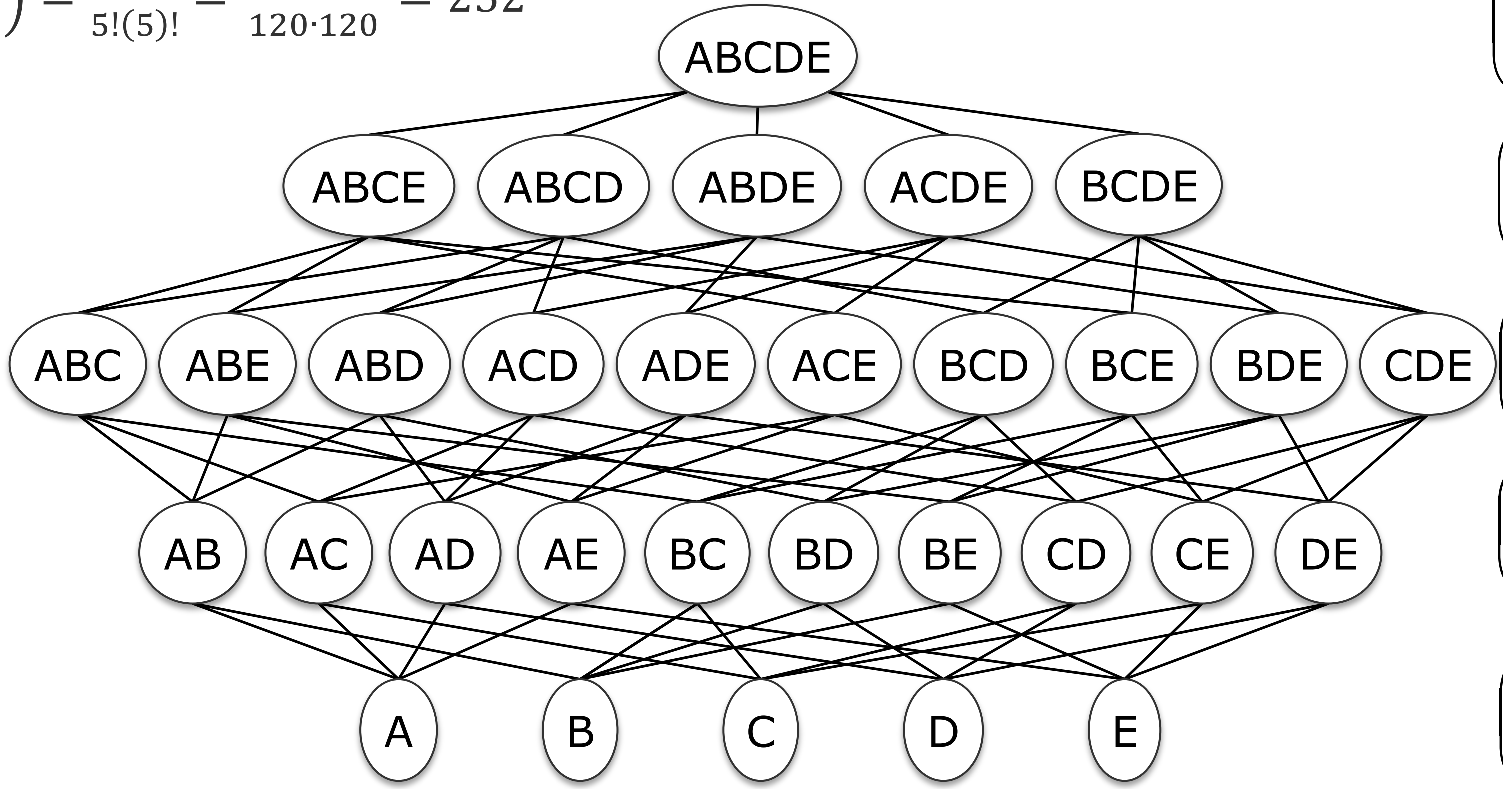
- Lattice
  - Partially ordered set (poset)
  - Each pair of elements has unique supremum and infimum



- Hasse Diagram
  - Drawing of partially ordered set
  - Each element is node
  - Edges upward to smallest larger element
    - Upward: Arrows no longer necessary

Basic lattice

- Represents each combination of elements
- $\binom{n}{k} = \frac{n!}{k!(n-k)!}$
- $\binom{10}{5} = \frac{10!}{5!(5)!} = \frac{3628800}{120 \cdot 120} = 252$



$$\binom{5}{5} = 1$$

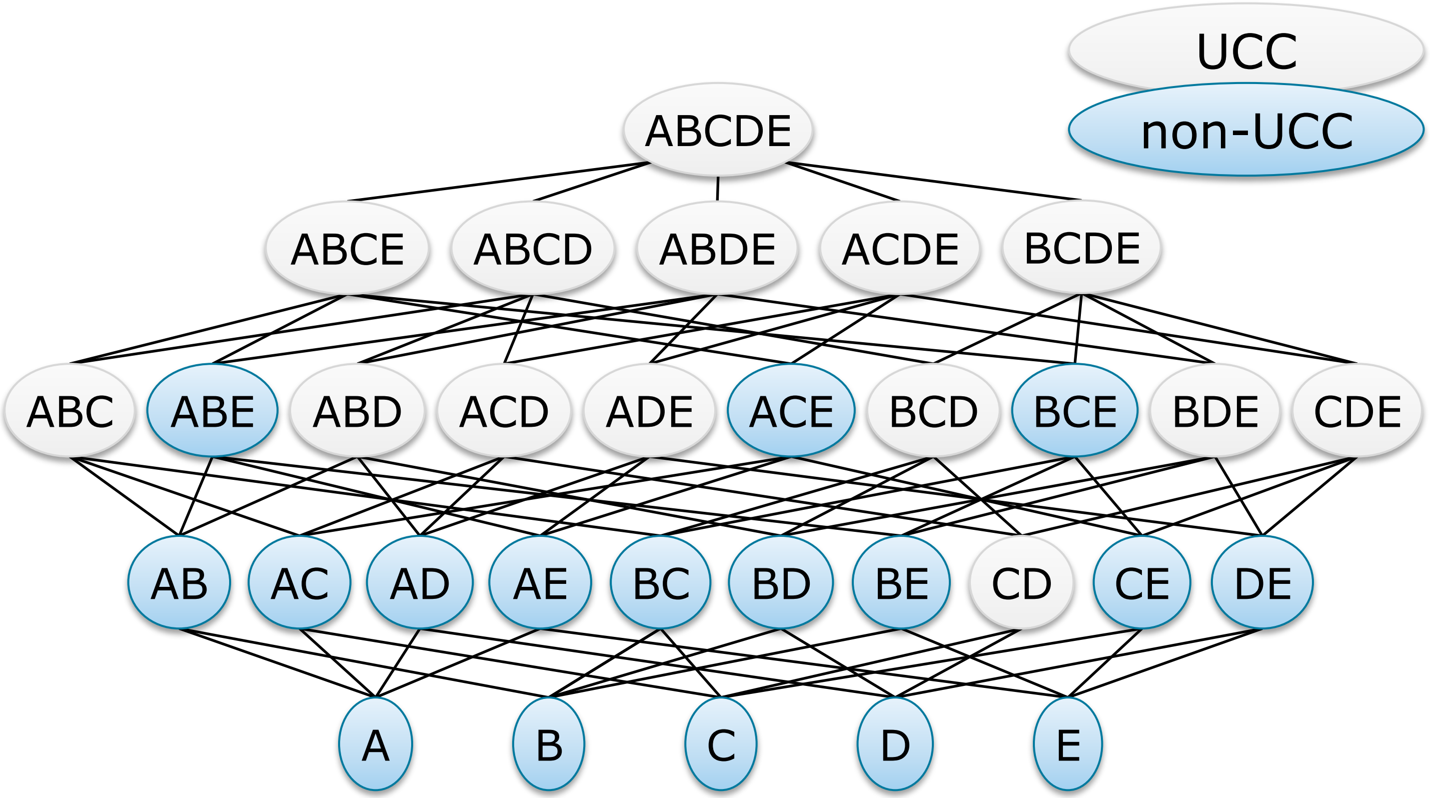
$$\binom{5}{4} = 5$$

$$\binom{5}{3} = \frac{5 \cdot 4}{2}$$

$$\binom{5}{2} = \frac{5 \cdot 4 \cdot 3}{2 \cdot 3}$$

$$\binom{5}{1} = \frac{5 \cdot 4 \cdot 3 \cdot 2}{2 \cdot 3 \cdot 4}$$

UCCs and non-UCCs in a Lattice

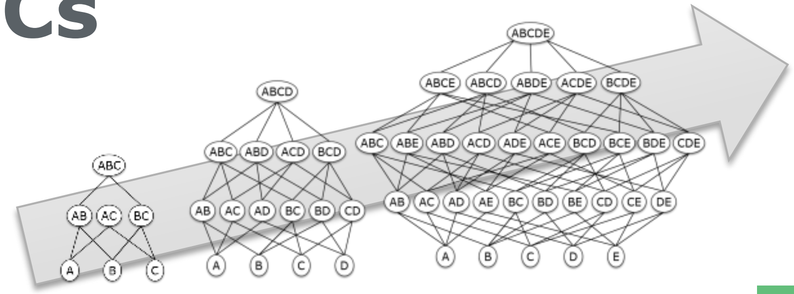


Prune BCD, BDE and CDE, because CD is unique

Candidate Set Growth for UCCs

HPI Hasso Plattner Institut

UCCs



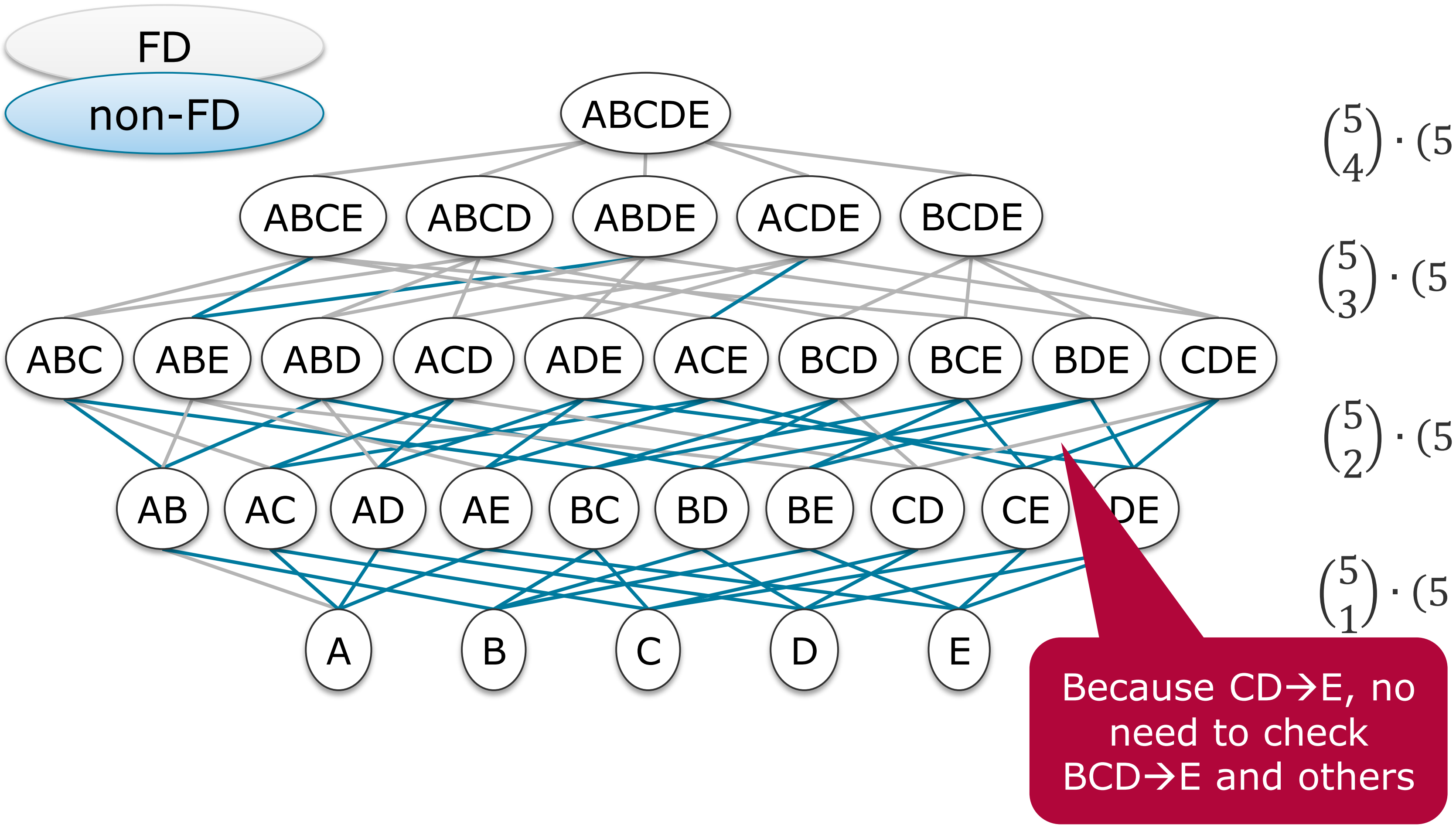
Number of levels: k

Total:	1	3	7	15	31	63	127	255	511	1,023	2,047	4,095	8,191	16,383	32,767
15															1
14														1	15
13													1	14	105
12												1	13	91	455
11											1	12	78	364	1,365
10										1	11	66	286	1,001	3,003
9									1	10	55	220	715	2,002	5,005
8								1	9	45	165	495	1,287	3,003	6,435
7							1	8	36	120	330	792	1,716	3,432	6,435
6						1	7	28	84	210	462	924	1,716	3,003	5,005
5					1	6	21	56	126	252	462	792	1,287	2,002	3,003
4				1	5	15	35	70	126	210	330	495	715	1,001	1,365
3			1	4	10	20	35	56	84	120	165	220	286	364	455
2		1	3	6	10	15	21	28	36	45	55	66	78	91	105
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Number of attributes: m

FDs and non-FDs in a Lattice

- Candidates for level *k* (lhs) and *m* attributes: $\binom{m}{k} \cdot (m-k)$



$\binom{5}{4} \cdot (5-4) = 5$

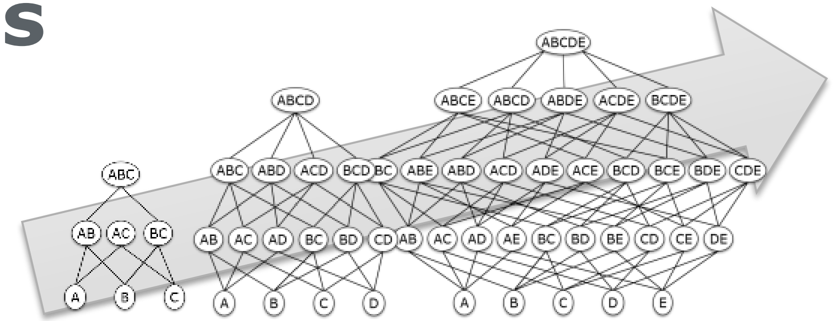
$\binom{5}{3} \cdot (5-3) = 20$

$\binom{5}{2} \cdot (5-2) = 30$

$\binom{5}{1} \cdot (5-1) = 20$

Candidate Set Growth for FDs

FDs

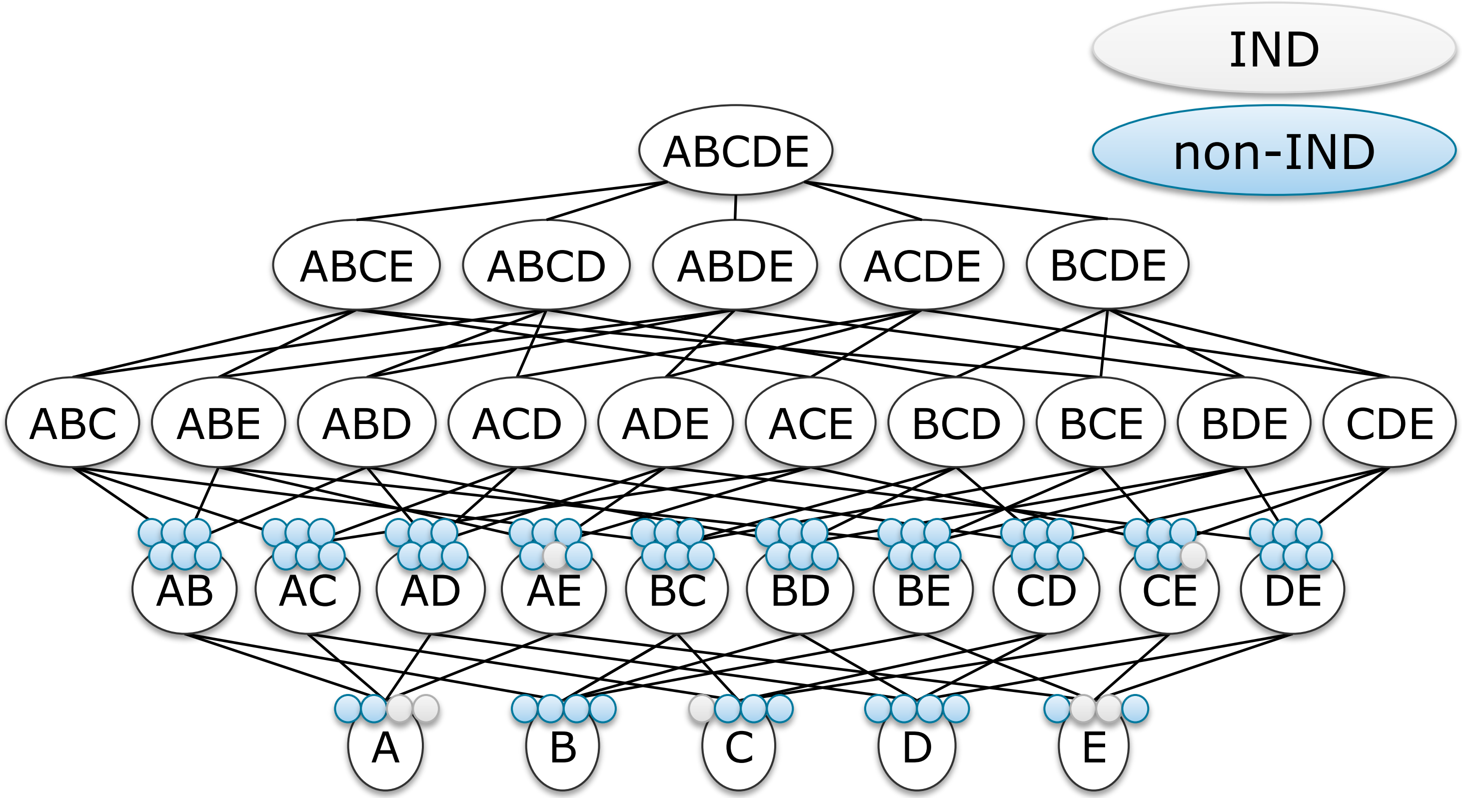


Total:	0	2	9	28	75	186	441	1,016	2,295	5,110	11,253	24,564	53,235	114,674	245,745
15															0
14														0	15
13													0	14	210
12												0	13	182	1,365
11											0	12	156	1,092	5,460
10										0	11	132	858	4,004	15,015
9									0	10	110	660	2,860	10,010	30,030
8								0	9	90	495	1,980	6,435	18,018	45,045
7							0	8	72	360	1,320	3,960	10,296	24,024	51,480
6						0	7	56	252	840	2,310	5,544	12,012	24,024	45,045
5					0	6	42	168	504	1,260	2,772	5,544	10,296	18,018	30,030
4				0	5	30	105	280	630	1,260	2,310	3,960	6,435	10,010	15,015
3			0	4	20	60	140	280	504	840	1,320	1,980	2,860	4,004	5,460
2		0	3	12	30	60	105	168	252	360	495	660	858	1,092	1,365
1	0	2	6	12	20	30	42	56	72	90	110	132	156	182	210
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Number of levels: k

Number of attributes: m

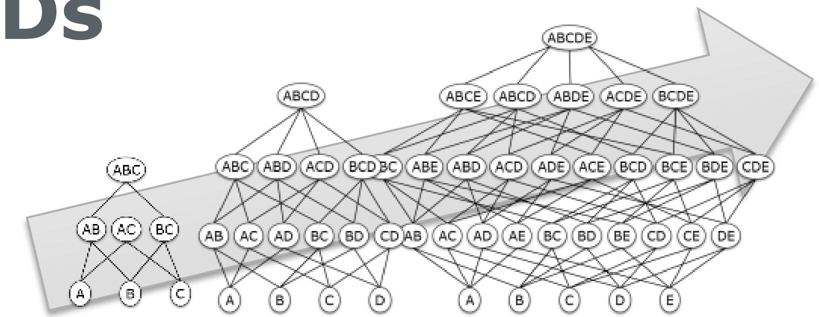
INDs and non-INDs in a Lattice



- For X⊆Y we assume here that X∩Y=∅. Other IND definitions are possible.
- INDs live only in bottom half of lattice

Candidate Set Growth for INDs

INDs



Number of levels: k															
Total:	0	2	6	24	80	330	1,302	5,936	26,784	133,650	669,350	3,609,672	19,674,096	113,525,594	664,400,310
15															0
14														0	0
13													0	0	0
12												0	0	0	0
11											0	0	0	0	0
10										0	0	0	0	0	0
9									0	0	0	0	0	0	0
8								0	0	0	0	0	0	0	0
7							0	0	0	0	0	0	0	17,297,280	259,459,200
6						0	0	0	0	0	0	665,280	8,648,640	60,540,480	302,702,400
5					0	0	0	0	0	30,240	332,640	1,995,840	8,648,640	30,270,240	90,810,720
4				0	0	0	0	1,680	15,120	75,600	277,200	831,600	2,162,160	5,045,040	10,810,800
3			0	0	0	120	840	3,360	10,080	25,200	55,440	110,880	205,920	360,360	600,600
2		0	0	12	60	180	420	840	1,512	2,520	3,960	5,940	8,580	12,012	16,380
1	0	2	6	12	20	30	42	56	72	90	110	132	156	182	210
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Number of attributes: m

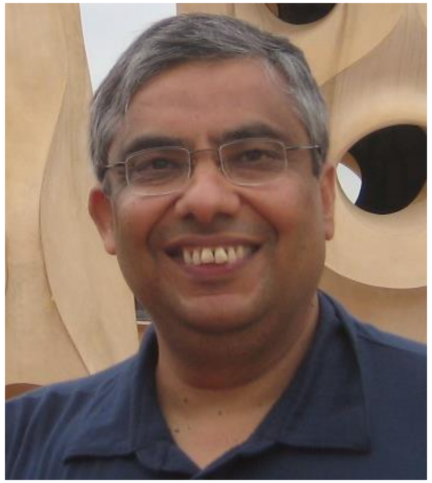
Overview

1. The lattice
2. **Apriori lattice traversal**
3. Position List Indices
4. Bloom filters

R. Agrawal, R. Srikant
"Fast Algorithms for Mining Association Rules"
Proc. of the Int'l Conference on Very Large Databases (VLDB), 1994

The Authors

- Rakesh Agrawal
  - 1983 Ph.D. at University of Wisconsin, Madison
  - 1983-1989 Bell Laboratories
  - 1989-2006 IBM Research
  - 2006-2014 Microsoft Research
  - Data Mining Pioneer
- Ramakrishnan Srikant
  - Ph.D. at University of Wisconsin, Madison
  - IBM Almaden Research Center
  - Since 2006 Distinguished Scientist at Google



Slides partly based on talk by Felix Leupold

Terminology

- Itemset I, Transactionset D
- Association rule
  - Statement of the form X→Y, where $X \subset I$, $Y \subset I$ and $X \cap Y = \{ \}$
- Support (frequency of a subset):
  - In what percentage of transactions is $X \cup Y$ present?
  - → minSup
- Confidence (of rule):
  - In what percentage of transactions in which X appears does Y also appear?
  - → minConf
- Algorithm
  1. Find all itemsets with minSup (large itemsets).
  2. From these, derive association rules with minConf.

Example

- Products:
  - I={Cola, Saft, Bier, Wein, Wasser, Schokolade, Brot, Schinken, Chips}
- Transactions T1 - T9:
  - T1={Saft, Cola, Bier}
  - T2={Saft, Cola, Wein}
  - T3={Saft, Wasser}
  - T4={Saft, Cola, Bier, Wein}
  - T5={Wasser}
  - T6={Schokolade, Cola, Chips}
  - T7={Cola, Bier}
  - T8={Schokolade, Schinken, Brot}
  - T9={Brot, Bier}
- minSup = 2

Key idea

- Problem
  - Many candidates (all subsets)
  - Checking transactions is expensive

- Idea: Support monotonically decreases with larger itemsets.
  - If a subset of some set M is small, then M itself is also small (= not large)
  1. Generate candidates using already discovered large itemsets.
  2. Delete all candidate that contain non-large subsets.
  - This suggests a bottom-up candidate generation approach

Apriori

$L_1 = \{large\ 1\text{-}itemsets\}$

For ($k = 2; L_{k\text{-}1} \neq \phi\ ; k{+}{+}$) do begin

  $C_k = \text{apriori-gen}(L_{k\text{-}1});$

  forall transactions $t \in D$ do begin

    $C_t = \text{subset}(C_k, t)$

    forall candidates $c \in C_t$ do

      $c.count{+}{+};$

    end

  end

  $L_k = \{\ c \in C_k \mid c.count \geq minsup\}$

end

$Answer = \bigcup_k L_k;$

Apriori-Gen

- 2 Steps:

1. „Join"

$$\begin{array}{l}\textbf{insert into } C_k \\ \textbf{select } p.\text{item}_1,\ p.\text{item}_2,\ \ldots,\ p.\text{item}_{k-1},\ q.\text{item}_{k-1} \\ \textbf{from } L_{k-1}\ p,\ L_{k-1}\ q \\ \textbf{where } p.\text{item}_1 = q.\text{item}_1,\ \ldots,\ p.\text{item}_{k-2} = q.\text{item}_{k-2}, \\ \qquad p.\text{item}_{k-1} < q.\text{item}_{k-1};\end{array}$$

p and q are identical in the first *k*-2 items

2. „Prune"

$$\begin{array}{l}\textbf{forall } \text{itemsets } c \in C_k \textbf{ do} \\ \quad \textbf{forall } (k-1)\text{-subsets } s \text{ of } c \textbf{ do} \\ \qquad \textbf{if } (s \notin L_{k-1}) \textbf{ then} \\ \qquad\quad \textbf{delete } c \text{ from } C_k;\end{array}$$

Delete all candidates with a subset not in L_{k-1}

Example

- T1={Saft, Cola, Bier}
- T2={Saft, Cola, Wein}
- T3={Saft, Wasser}
- T4={Saft, Cola, Bier, Wein}
- T5={Wasser}
- T6={Schokolade, Cola, Chips}
- T7={Cola, Bier}
- T8={Schokolade, Schinken, Brot}
- T9={Brot, Bier}

- minSup = 2

- Let L2= {{Cola,Saft}, {Cola,Bier}, {Cola,Wein}, {Saft,Bier}, {Saft,Wein}}

- Join: {Cola,Saft,Bier}, {Cola,Saft,Wein}, {Cola,Bier,Wein}, {Saft,Bier,Wein}

- Prune deletes: {Cola,Bier,Wein} and {Saft,Bier,Wein}

Creation of rules (not relevant for lattice traversal)

- Let L be a large itemset and A $\subseteq$ L

$$A \rightarrow (L - A) \quad \leftrightarrow \quad \frac{\sup(L)}{\sup(A)} > \min Conf$$

- Idea: If XY → Z not true, then X → YZ is also not true
  - Because sup(XY) ≤ sup(X)

- Again inductive generation (over number of elements on rhs)
  - Check all rules with on element on rhs
  - From these, generate 2-element rhs‘s and check these
  - ...

Extensions

- Consider quantity of items within transaction
- Consider order of transactions
- Consider taxonomy / catagories
  - Outerwear → Hiking Boots
- Remove useless/non-actionable rules for pruning
- Determine interesting of discovered rules

Overview

1. The lattice
2. Apriori lattice traversal
3. **Position List Indices**
4. Bloom filters

PLI Motivation

- UCC detection and FD detection need to know groups of rows with same value
  - UCCs: For a given attribute or attribute combination, are there ANY groups of such rows?
    - I.e., duplicates
  - FDs: For any such group (LHS), are there any dependent attributes (RHS) with also same values

- PLI for an attribute or set of attributes is a compact representation of such groups
  - Insight 1: Actual values are not needed, only row-ids
  - Insight 2: Singleton groups are not needed, only groups of size ≥ 2
  - Insight 3: PLIs for attribute sets can be efficiently built based on PLIs of their subsets.

Position List Indices (PLIs)

	first	last
0	James	Smith
1	John	Smith
2	Robert	Johnson
3	John	Smith
4	John	Johnson
5	James	Williams

PLI: first
0, 5
1, 3, 4
2

PLI: last
0, 1, 3
2, 4
5

Example by Jakob Zwiener

Position List Indices (PLIs)

	first	last
0	James	Smith
1	John	Smith
2	Robert	Johnson
3	John	Smith
4	John	Johnson
5	James	Williams

PLI: first
0, 5
1, 3, 4

PLI: last
0, 1, 3
2, 4

- If PLI is empty: Column is unique
- Next step: Determine PLI for larger attribute sets
  - Idea: PLI intersection = intersection for each pair of groups
  - If intersection is empty (after removing singletons), attribute combination is unique

Position List Indices (PLIs) – Intersection

	PLI first
1	0, 5
2	1, 3, 4

	Probing table
0	1
1	2
2	0
3	2
4	2
5	1

0 for singletons

	PLI last
1	0, 1, 3
2	2, 4

Value class	Row numbers
1, 1	0
2, 1	1, 3
2, 2	4

PLI first, last
1, 3

	Firstname	Lastname
0	James	Smith
1	**John**	**Smith**
2	Robert	Johnson
3	**John**	**Smith**
4	John	Johnson
5	Richard	Williams

Key Error

- Key-error of column or column combination: Number of records to be removed to that column becomes unique.
- $\mathrm{keyerror}(X) = \sum_{c \in pli(X)}(|c|) - |pli(X)|$
- If $\mathrm{keyerror}(X) = 0$, X is unique
- Trick later: Use key-error to infer FDs
  - TANE

	first	last
0	James	Smith
1	John	Smith
2	Robert	Johnson
3	John	Smith
4	John	Johnson
5	James	Williams
keyerror	3	3
keyerror	1	

Analysis of PLIs

- Intersection is associative
  - Potential to optimize intersection order
  - Build larger PLIs from smaller ones

- Intersection is commutative
  - Hash bigger PLI and probe smaller PLI
  - To reduce number of tests

- If there is enough main memory
  - Keep PLI of columns in main memory
  - Going up in the lattice requires only to probe the current PLI
    - Becomes increasingly fast when going up
    - <1ms for most combinations

- Going down the lattice
  - Unfortunately, PLIs do not help
  - Start from scratch

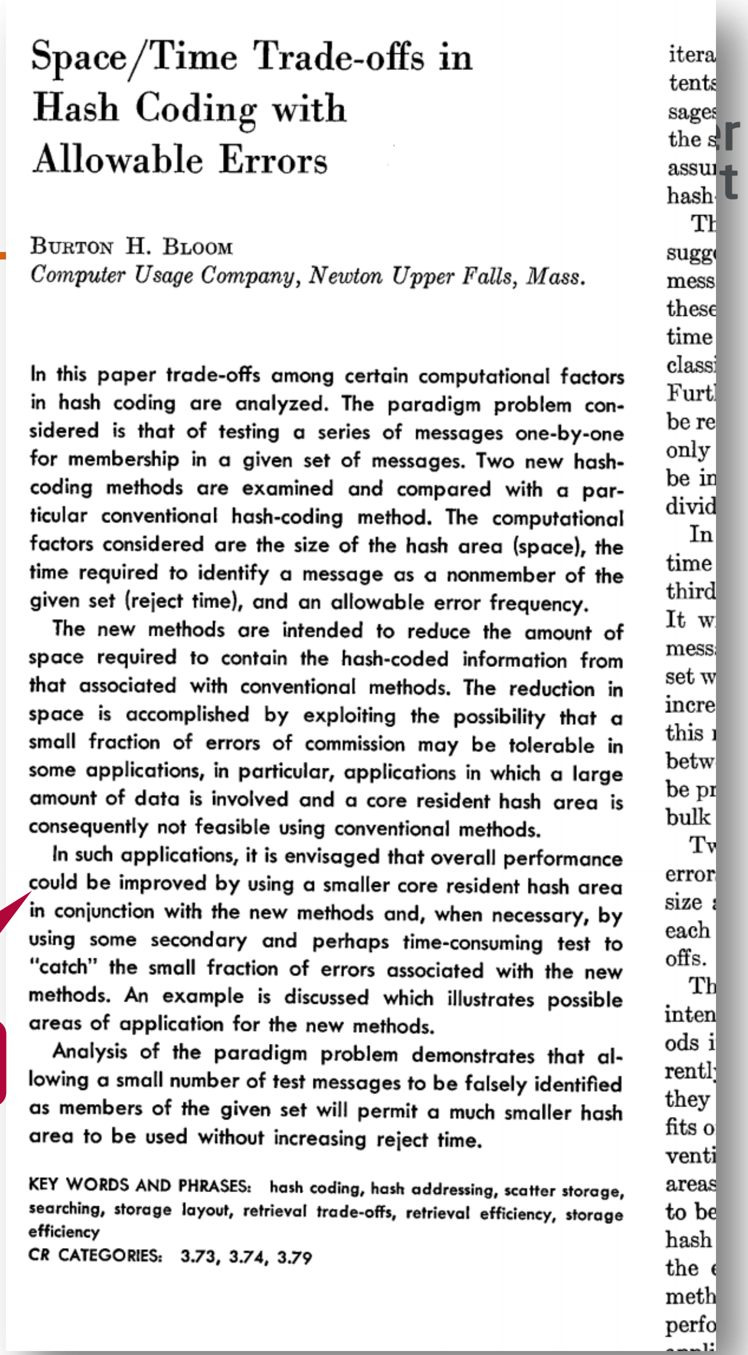
Overview

1. The lattice
2. Apriori lattice traversal
3. Position List Indices
4. **Bloom filters**

Searching for element (or confirming its existence)

- Problem: Is x an element of column A?
  - Column A is large – must be stored on disk
- Idea: Store small representation of A in main memory
- Bloom filter: Developed 1970 by Burton Howard Bloom
  - > 5.500 citations
- Build Boolean hash-table T on A using all available main memory.
- Use T to mark whether an element is k in A
- Test can fail, but only in one direction
  - If $k \in T$, we cannot be sure whether $k \in A$.
  - If $k \notin T$, we know that $k \notin A$.
- T acts as filter: Bloom-filter
- Improvement: Use j independent hash-function
  - Improves false positive rate

> 6,000 citations



Space/Time Trade-offs in Hash Coding with Allowable Errors

Burton H. Bloom
Computer Usage Company, Newton Upper Falls, Mass.

In this paper trade-offs among certain computational factors in hash coding are analyzed. The paradigm problem considered is that of testing a series of messages one-by-one for membership in a given set of messages. Two new hash-coding methods are examined and compared with a particular conventional hash-coding method. The computational factors considered are the size of the hash area (space), the time required to identify a message as a nonmember of the given set (reject time), and an allowable error frequency.

The new methods are intended to reduce the amount of space required to contain the hash-coded information from that associated with conventional methods. The reduction in space is accomplished by exploiting the possibility that a small fraction of errors of commission may be tolerable in some applications, in particular, applications in which a large amount of data is involved and a core resident hash area is consequently not feasible using conventional methods.

In such applications, it is envisaged that overall performance could be improved by using a smaller core resident hash area in conjunction with the new methods and, when necessary, by using some secondary and perhaps time-consuming test to "catch" the small fraction of errors associated with the new methods. An example is discussed which illustrates possible areas of application for the new methods.

Analysis of the paradigm problem demonstrates that allowing a small number of test messages to be falsely identified as members of the given set will permit a much smaller hash area to be used without increasing reject time.

KEY WORDS AND PHRASES: hash coding, hash addressing, scatter storage, searching, storage layout, retrieval trade-offs, retrieval efficiency, storage efficiency
CR CATEGORIES: 3.73, 3.74, 3.79