Unique column combinations

Arvid Heise

Guest lecture in Data Profiling and Data Cleansing
Prof. Dr. Felix Naumann

Agenda

Introduction and problem statement

- Unique column combinations
- Exponential search space
- Null values
- General pruning techniques

Discovery algorithms

- Apriori
- HCA
- DUCC
- Gordian

Agenda

Introduction and problem statement

- Unique column combinations
- Exponential search space
- Null values
- General pruning techniques

Discovery algorithms

- Apriori
- HCA
- DUCC
- Gordian

Unique column combinations

- Relational model
  - Dataset R with schema S
- Unique column combination $K \subseteq S$

$$\forall r_i, r_j \in R : i \neq j \Rightarrow r_i[K] \neq r_j[K]$$

- In the following, they are called uniques
- Examples: all primary keys, all unique constraints

A	B	C
a	1	x
b	2	x
c	2	y

  - Uniques: {A, AB, AC, BC, ABC}
  - Non-uniques: {B, C}

Minimal uniques

- We are mostly interested in minimal uniques $K \subseteq S$

$$\neg \exists K' \subseteq S : unique(K') \wedge K' \subset K$$

- Removal of any column leads to non-unique combination

- For the previous example: {A, BC}
- Redundant: {AB, AC, ABC}

A	B	C
a	1	x
b	2	x
c	2	y

- Candidates for primary keys

Maximal non-uniques

- Analogously we can define maximal non-uniques $K \subseteq S$

$$\neg \exists K' \subseteq S : non-unique(K') \wedge K \subset K'$$

- Adding any column leads to unique combination

- Non-unique: {AB, AC}
- Redundant: {A, B, C}

A	B	C
a	1	x
a	2	x
a	2	y

- May be a data quality problem

Applications

- Learning characteristics about a new data set
- Database management
  - Finding a primary key
  - Finding unique constraints
- Query optimization
  - Cardinality estimations for joins
- Finding duplicates / data quality issues
  - If expected unique column combinations are not unique
  - Or with approximate uniques

Agenda

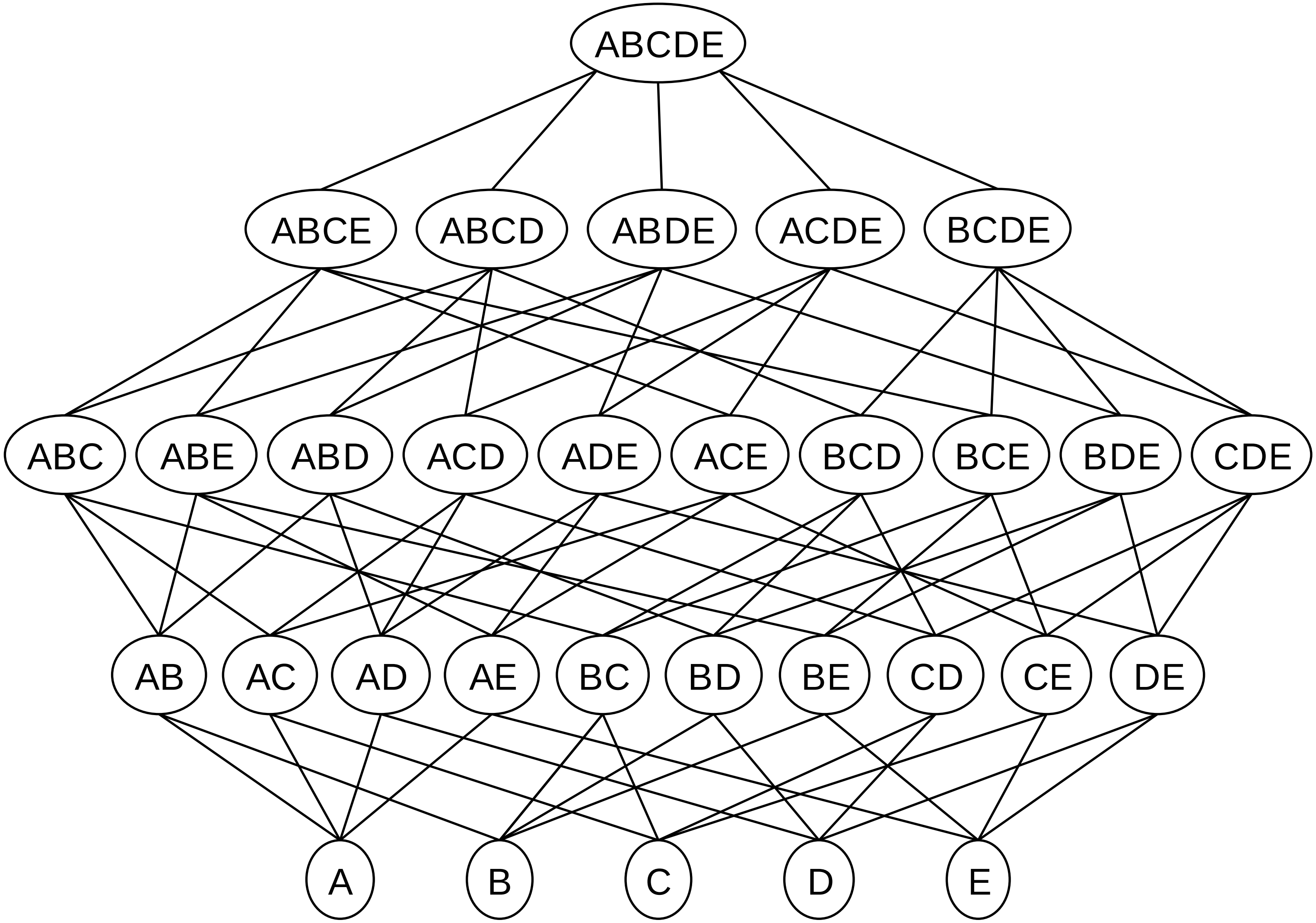
Introduction and problem statement

- Unique column combinations
- Exponential search space
- Null values
- General pruning techniques

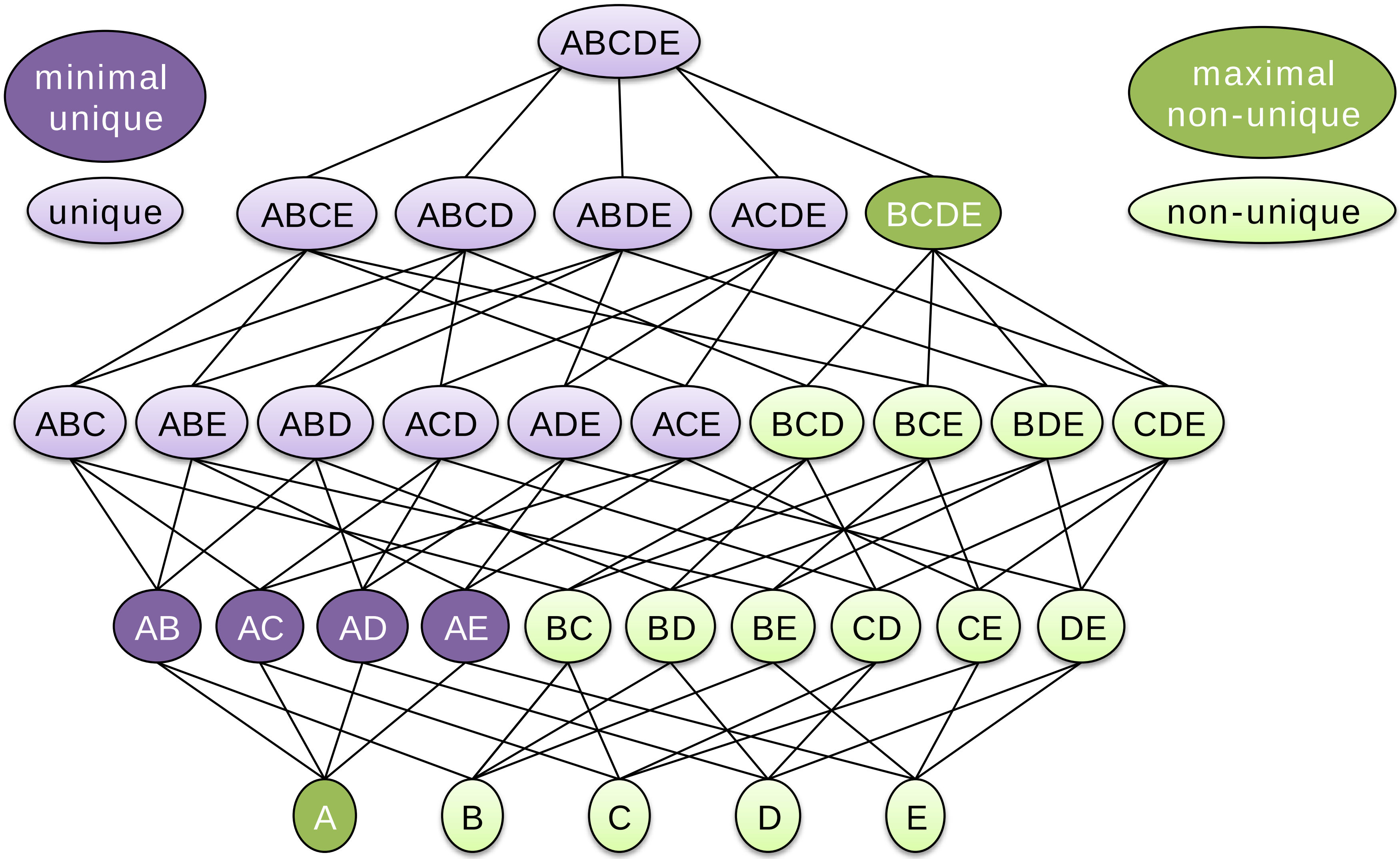
Discovery algorithms

- Apriori
- HCA
- DUCC
- Gordian

Exponential search space



Result of algorithm

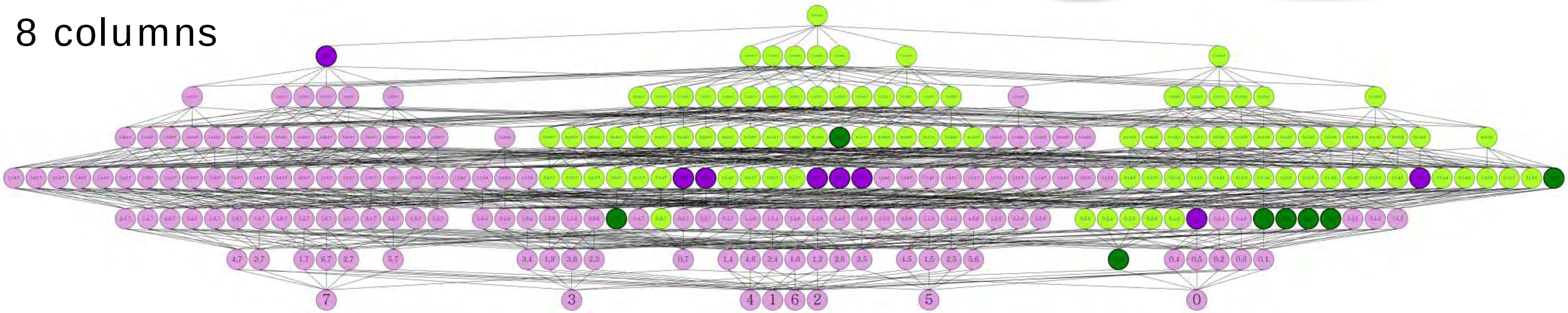


TPCH line item

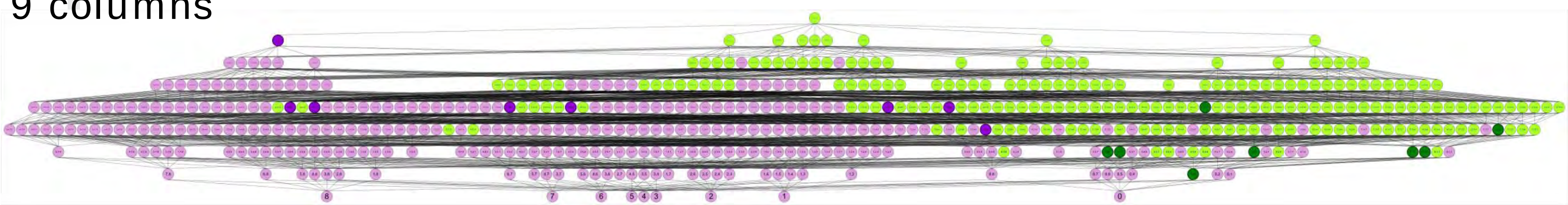
unique

non-unique

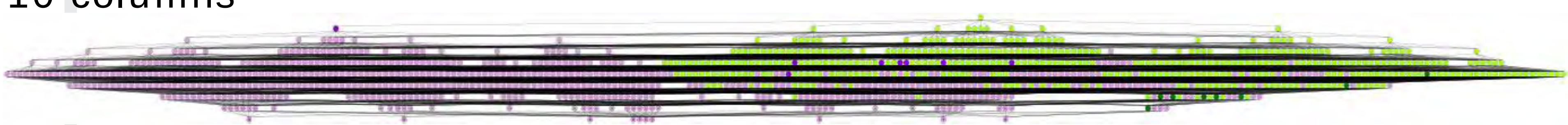
8 columns



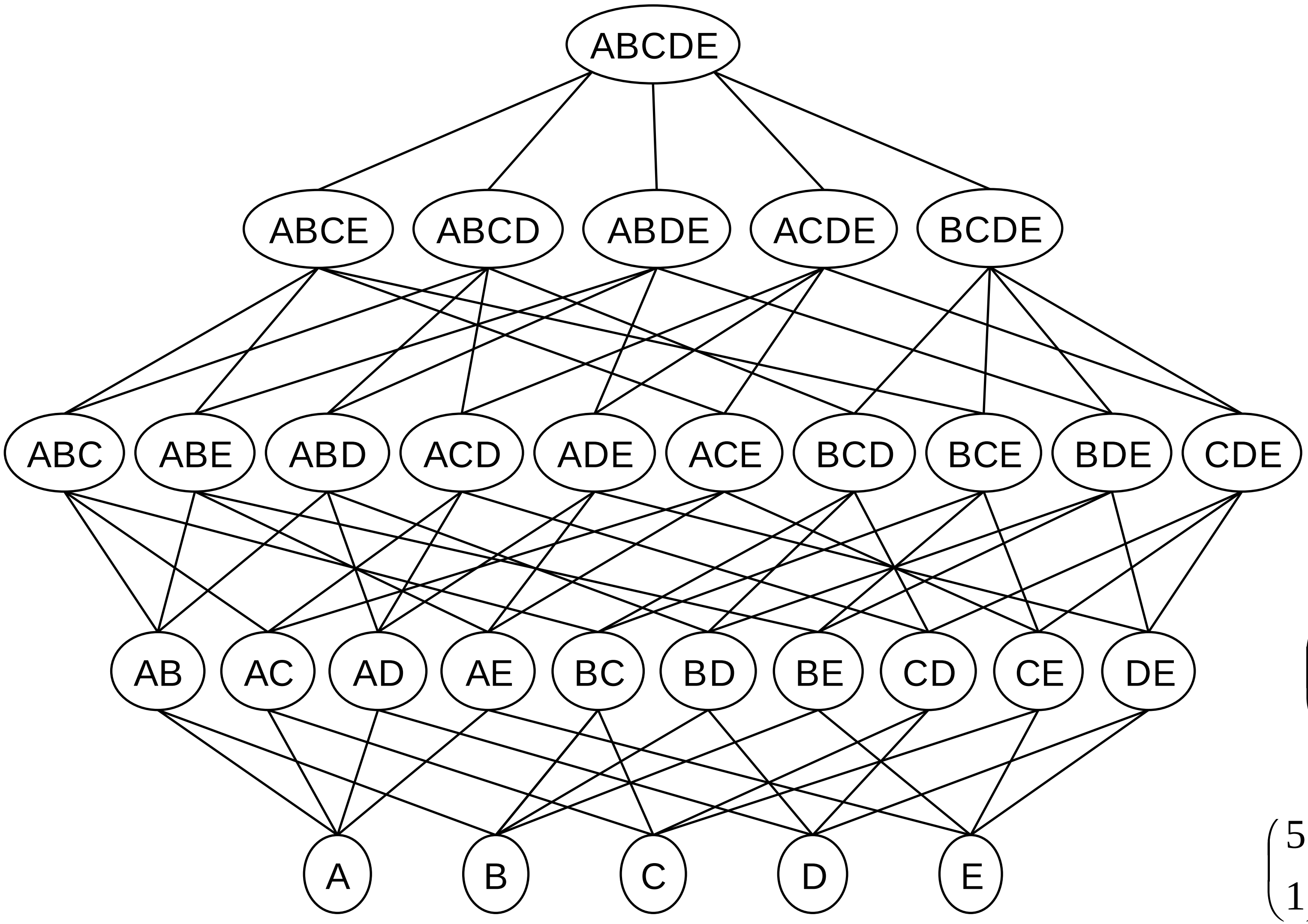
9 columns



10 columns



Size of the lattice



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

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

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

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

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

Computational feasibility

For a lattice over n columns

- $\binom{n}{k}$ combinations of size k
- All combinations: 2^n-1 (let's ignore -1 for the remaining slides)
- Largest solution set: $\binom{n}{n/2}$ minimal uniques are of size $\frac{n}{2}$
  - Verifying minimality, requires to check also all combinations of size $\frac{n}{2} - 1$
- Adding a column doubles search space

Brute forcing Uniprot

- Data set about proteins with 223 columns

- Combinations: ~$1.3*10^{67}$
- Largest solution: ~$7.2*10^{65}$
  - There are roughly 10^{50} atoms on earth

- Assuming all uniques are of size 1-9
  - $\binom{223}{9} + \binom{223}{8} + \ldots \approx 3.3 \cdot 10^{15}$
  - 1ms verification time results in 100ka processing time

Agenda

Introduction and problem statement

- Unique column combinations
- Exponential search space
- Null values
- General pruning techniques

Discovery algorithms

- Apriori
- HCA
- DUCC
- Gordian

Null values

- Null values have a wide range of interpretations
  - Unknown (birth day)
  - Non-applicable (driver license number for kids)
  - Undefined (result of integration/outer join)

- What is the minimal unique for the following data set?

A	B	C	D
a	1	x	1
b	2	y	2
c	3	z	5
d	3	⊥	5
e	⊥	⊥	5

Handling null values #1

- Depends on the actual application

- To find primary keys
  - Remove all columns with null values
  - Result: {A}

A	B	C	D
a	1	x	1
b	2	y	2
c	3	z	5
d	3	⊥	5
e	⊥	⊥	5

Handling null values #2

- Depends on the actual application

- To define unique constraints
  - SQL defines grouping for null: null != null
  - Result: {A, C} -> CD unique
  - A column of nulls is unique!

A	B	C	D
a	1	x	1
b	2	y	2
c	3	z	5
d	3	⊥	5
e	⊥	⊥	5

Handling null values #3

- Depends on the actual application
- To define unique constraints
  - SQL defines distinctness for null: null = null
  - Result: {A, BC}

A	B	C	D
a	1	x	1
b	2	y	2
c	3	z	5
d	3	⊥	5
e	⊥	⊥	5

Agenda

Introduction and problem statement

- Unique column combinations
- Exponential search space
- Null values
- General pruning techniques

Discovery algorithms

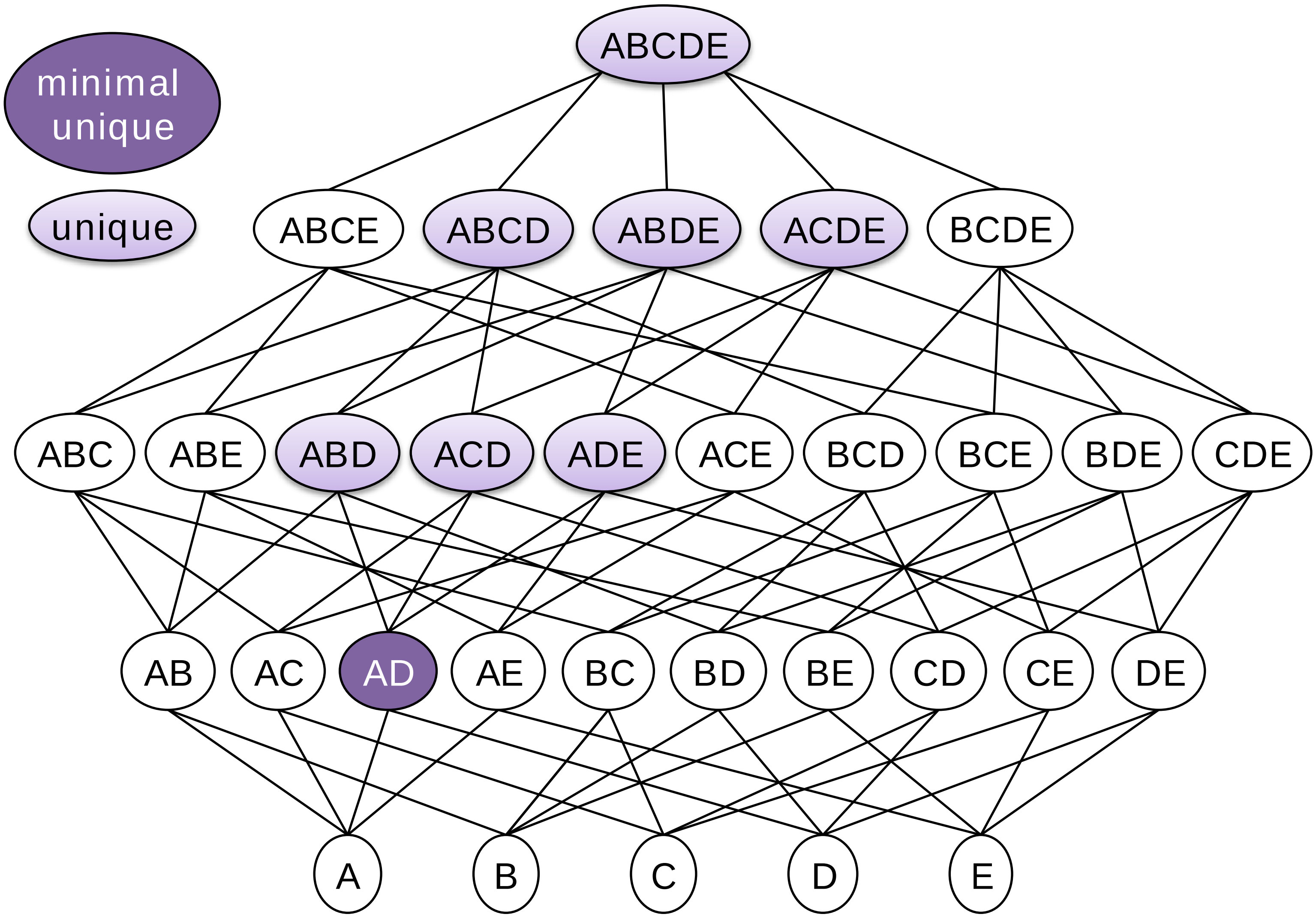
- Apriori
- HCA
- DUCC
- Gordian

Pruning with uniques

- Pruning: inferring the type of a combination without actual verification

- If A is unique, supersets must be unique

Pruning effect of a pair



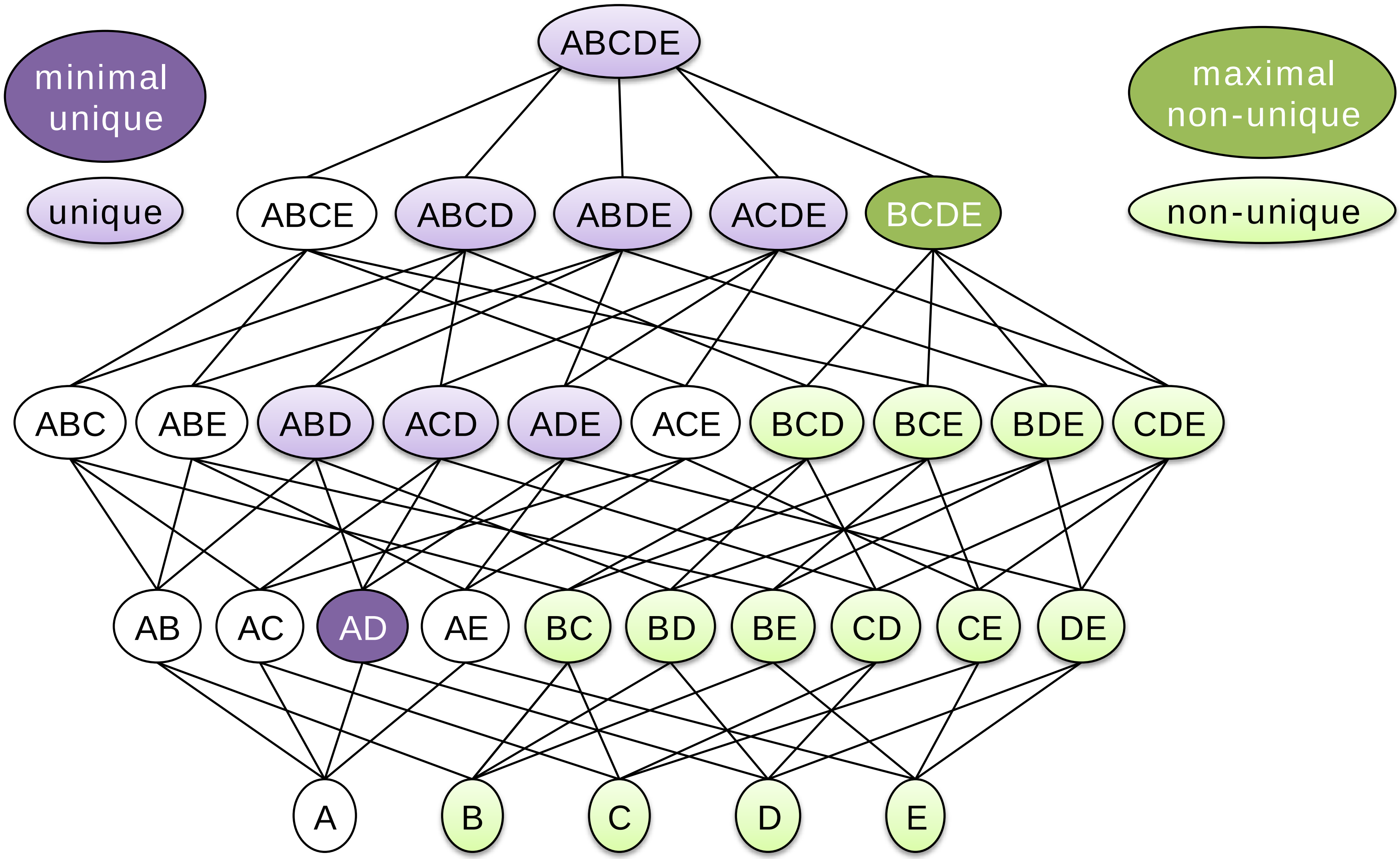
Pruning with uniques #2

- Pruning: inferring the type of a combination without actual verification

- If A is unique, supersets must be unique
- Finding a unique column prunes half of the lattice
  - Remove column from initial data set and restart
- Finding a unique column pair removes a quarter of the lattice
  - In general, the lattice over the combination is removed

- The pruning power of a combination is reduced by prior findings
  - AB prunes a quarter
  - BC additionally prunes only one eighth
  - ABC already pruned one eights

Pruning both ways



Pruning on-the-fly

- Materialization of the lattice is infeasible
  - Only possible for few columns
  - Nodes cannot be removed when discovering unique

- Prune on-the-fly
  - Enumerate nodes as before
  - Skip a node that has been pruned
  - Depending on the approach that might be challenging
  - Might require an efficient index structure
  - Often: candidate generation

Agenda

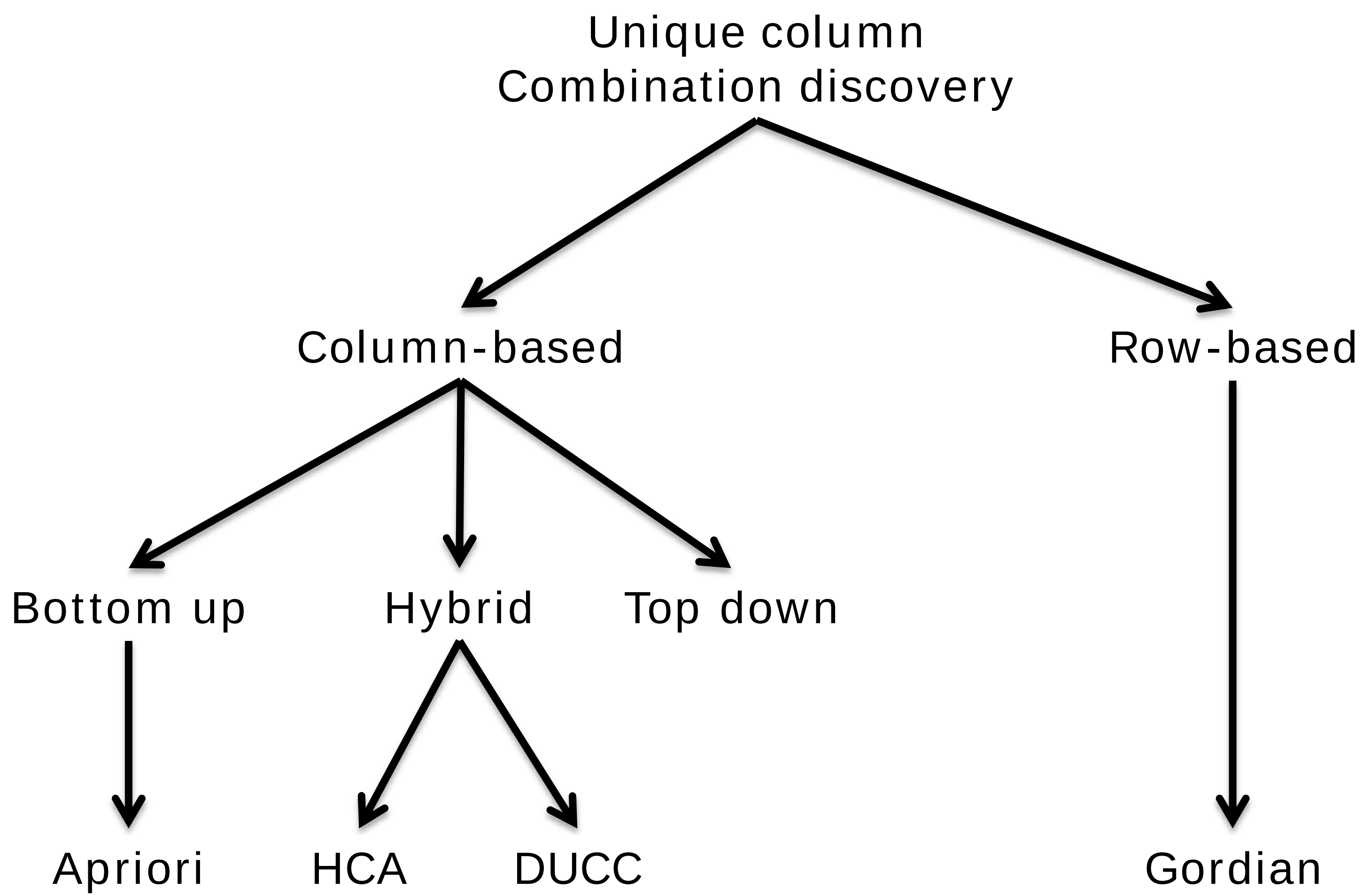
Introduction and problem statement

- Unique column combinations
- Exponential search space
- Null values
- General pruning techniques

Discovery algorithms

- Apriori
- HCA
- DUCC
- Gordian

Discovery Algorithms



Column-based algorithms

- Traverse through lattice

- Check for uniqueness
  - Different approaches possible
  - Use database back end and distinctness query
    - SELECT COUNT(DISTINCT A, B, C) FROM R
    - Compare with row number
  - Position list indexes (explained later)
  - For now, check is blackbox

- Prune lattice accordingly

Agenda

Introduction and problem statement

- Unique column combinations
- Exponential search space
- Null values
- General pruning techniques

Discovery algorithms

- Apriori
- HCA
- DUCC
- Gordian

Apriori-based

- C. Giannella and C. M. Wyss. "Finding minimal keys in a relation instance." (1999).
- Actually does not use much of the apriori idea

- Basic idea:
  - Using the state of combinations of size k
  - We need to visit only unpruned combinations of size k+1

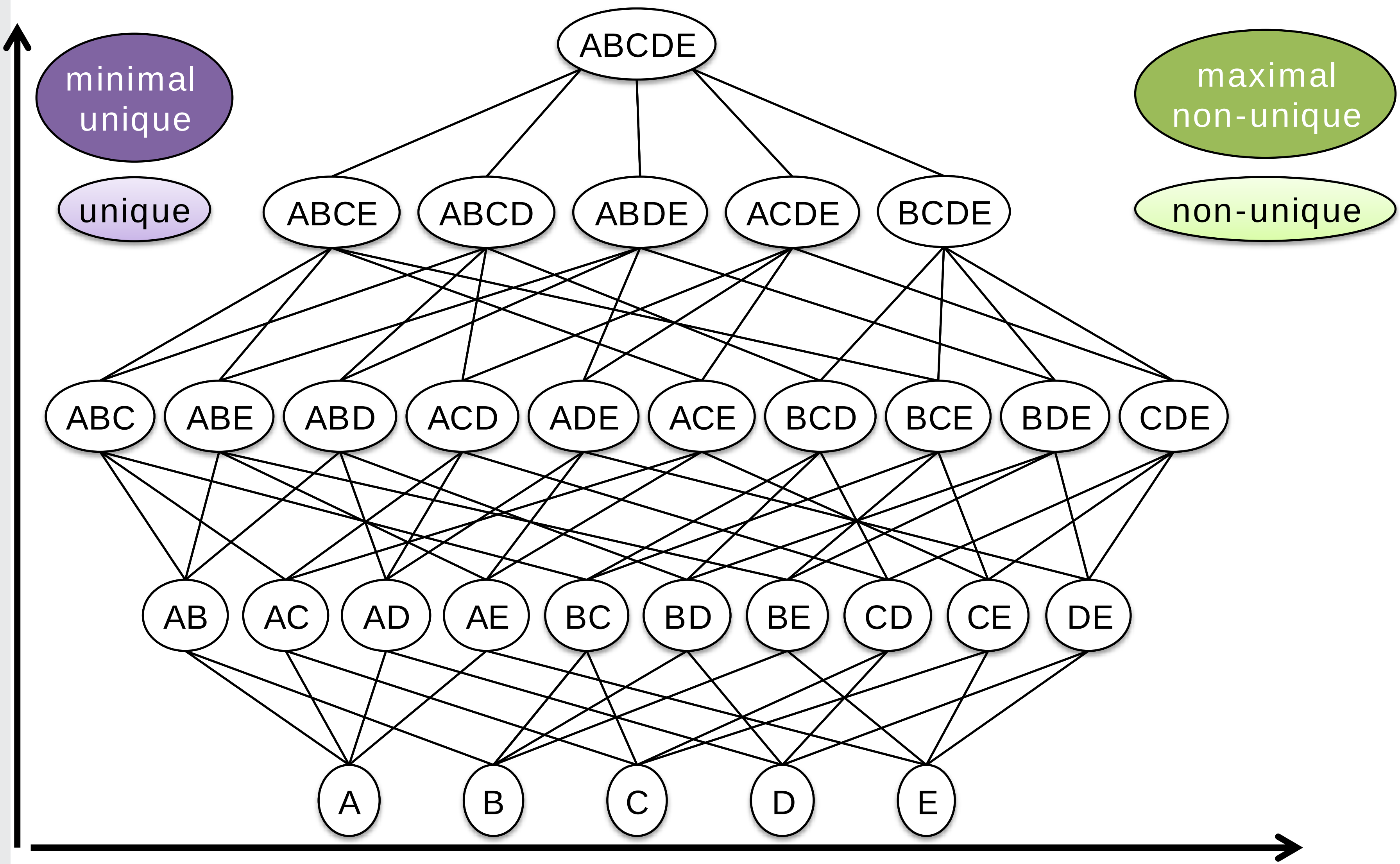
- Start with columns
- Check pairs of non-unique columns
- Check triples of non-unique pairs ...
- Terminate if no new combinations can be enumerated

Candidate generation

- Do not generate too many duplicate combinations
- ABC, ABD, ACD, and BCD could point to ABCD

- Apriori: prefix-based generation
- Generate only combination of size n if prefix n-1 matches
- Only ABC and ABD can generate ABCD
  - Still redundant verifications

Apriori visualized



Characteristics of Apriori

- Works well for small uniques
  - Bottom-up checks columns first
- Best case: all columns are unique
  - n checks
- Worst case: no uniques = one duplicate row
  - 2^n checks
- Apriori is exponential to n

Extensions

Top-down

- Start from top and go down
- Performs better if solution set is high up
- Candidate pruning becomes more tricky

Hybrid

- Combine bottom-up and top-down
- Interleave checks
- Works well if solution set has many small and large columns
- Worst case: solution set in the middle

Agenda

Introduction and problem statement

- Unique column combinations
- Exponential search space
- Null values
- General pruning techniques

Discovery algorithms

- Apriori
- HCA
- DUCC
- Gordian

Histogram-Count-based Apriori

- Ziawasch Abedjan and Felix Naumann. "Advancing the discovery of unique column combinations." *Proceedings of the international Conference on Information and Knowledge Management*. 2011.
- Extension of bottom-up apriori

- More sophisticated candidate generation
- Uses histograms for pruning
- Finds and uses functional dependencies on-the-fly

HCA candidate generation

- Maintains a sorted list of non-uniques
  - Avoids duplicate generation of combinations
- Prunes non-minimal uniques efficiently
  - ABC unique, ABD is non-unique
  - ABD would generate ABCD
  - HCA performs quick minimality check with bitsets
- Hybrid approach
  - At least checks if remaining columns contains duplicates

Statistics

- Prunes column combinations that cannot be unique
  - A and B contains the same value for 4/7 of the data
  - C contains the same value for 5/7 of the data
  - AC cannot be unique, AB might (not very likely)
- Especially viable if there are already indices

A	B	C
1	A	U
1	A	U
1	A	U
1	A	U
2	B	U
3	C	V
4	D	W

Functional dependencies

- Functional dependency
  - Value of one column determines value of another
  - Birthday->age

- Intuition:
  - If A->B and A non-unique, B must be non-unique
  - If A->B and B unique, A must also be unique
  - If A->B and AC non-unique, BC must be non-unique
  - If A->B and BC unique, AC must also be unique

- FD A->B can be found with histogram of AB and B
  - Histograms of FDs have the same distinctness counts

Analysis of HCA

- Works well on data sets with small numbers of columns
- Quickly converges for many small combinations
  - Efficient pruning
- Saves distinctness checks for many pairs

- For larger combinations statistics become less important
  - At some point has to try all combinations
- Suffers from the same general complexity of Apriori

Agenda

Introduction and problem statement

- Unique column combinations
- Exponential search space
- Null values
- General pruning techniques

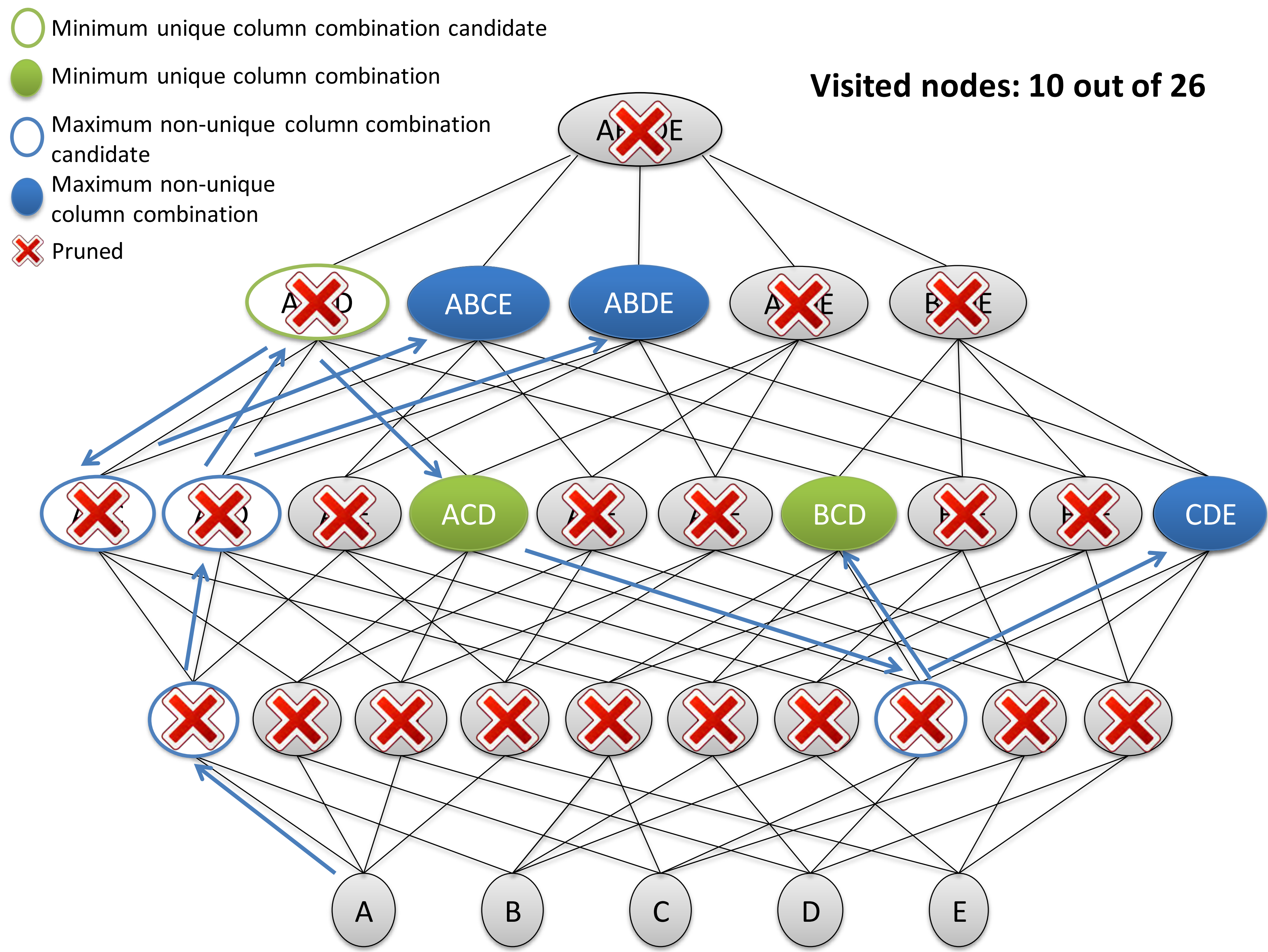
Discovery algorithms

- Apriori
- HCA
- DUCC
- Gordian

DUCC

- Arvid Heise, Jorge-Arnulfo Quiané-Ruiz, Ziawasch Abedjan, Anja Jentzsch, and Felix Naumann, "Scalable Discovery of Unique Column Combinations", *in preparation*
- Done during internship at QCRI

- Basic idea: random walk through lattice
- Pick random superset if current combination is non-unique
- Pick random subset otherwise
- Lazy prune with previously visited nodes



Minimum unique column combination candidate
Minimum unique column combination
Maximum non-unique column combination candidate
Maximum non-unique column combination
Pruned
Visited nodes: 10 out of 26
ABCE
ABDE
ACD
BCD
CDE
A
B
C
D
E

Position List Index

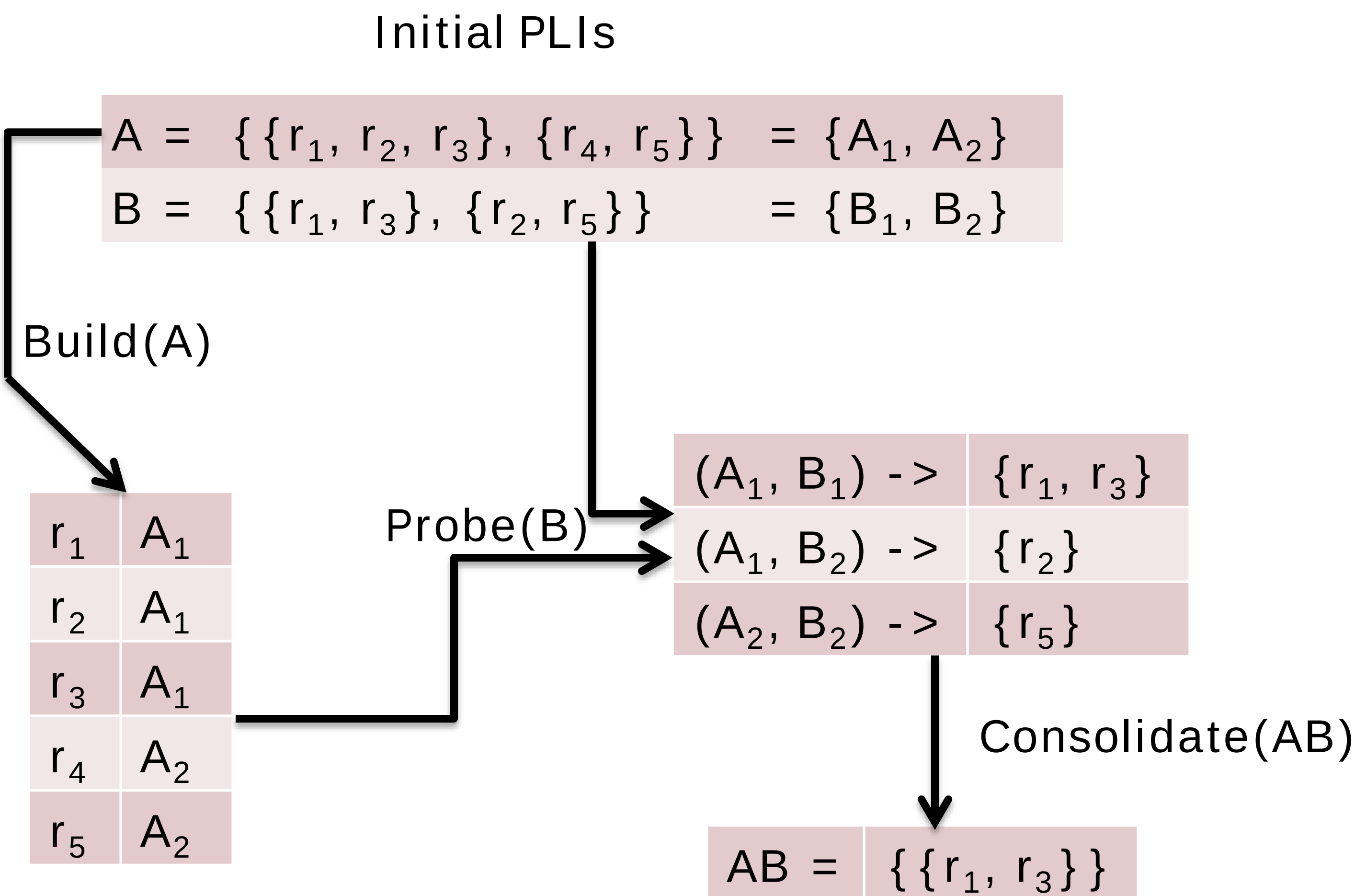
- Incorporates row-based pruning
- Intuition: number of duplicates decrease when going up
  - Many unnecessary rows are checked again and again

- Keep track of duplicates with inverted index
  - A: a->{r_1, r_2, r_3}, b->{r_4, r_5}
  - B: 1->{r_1, r_3}, 2->{r_2, r_5}

- We don't need the actual value
  - A: {{r_1, r_2, r_3}, {r_4, r_5}}
  - B: {{r_1, r_3}, {r_2, r_5}}

A	B
a	1
a	2
a	1
b	3
b	2

PLI Intersection



Analysis of PLI

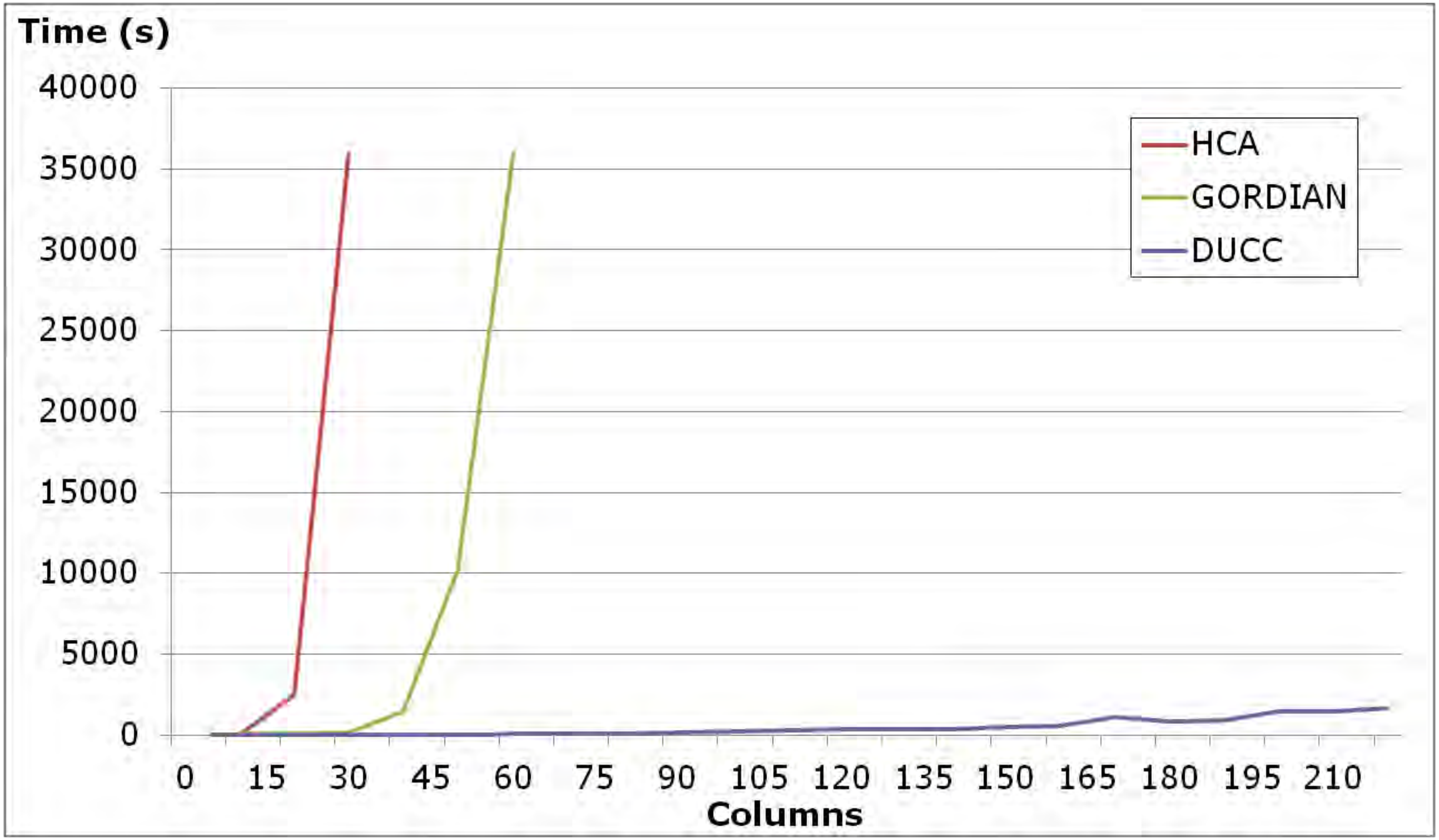
- Space complexity: $n \cdot \text{sizeof(long)} + \frac{n}{2} \cdot \text{sizeof(array)}$
- Intersection time complexity: $O(n+n)$

- Hash bigger PLI and probe smaller PLI
- 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
  - Unfortunately, PLI does not help
  - Start from scratch

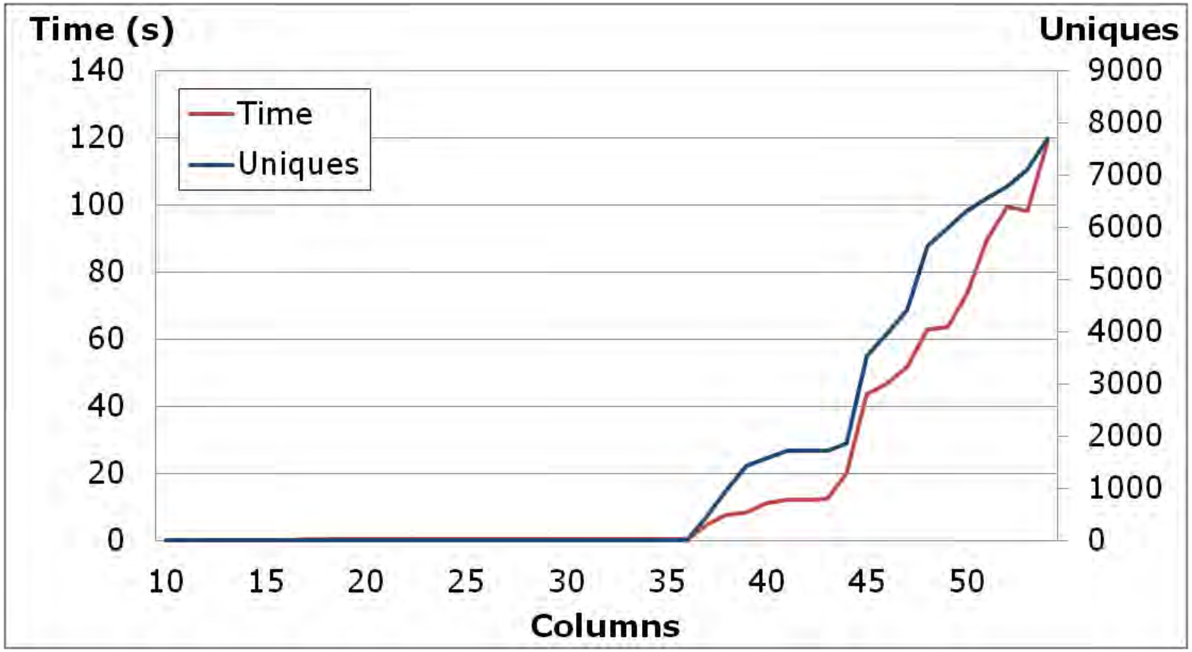
Experiments

- Uniprot, 100k rows, (DUCC null = null)



Analysis of DUCC

- Runtime mainly depends on size of solution set



- Worst case: solution set in the middle
- Aggressive pruning may lead to loss of minimal uniques!
  - Gordian’s final step can be used to plug these holes

Scaling up and out

- Scalability is major design goal of DUCC
- Random walk well suited for parallelization
  - Few coordination overhead

- Threads/worker share findings through event bus
  - Uniques/non-uniques
  - Holes in graph

- Lock-free to avoid bottlenecks
  - Only memory barrier in local event bus

Agenda

Introduction and problem statement

- Unique column combinations
- Exponential search space
- Null values
- General pruning techniques

Discovery algorithms

- Apriori
- HCA
- DUCC
- Gordian

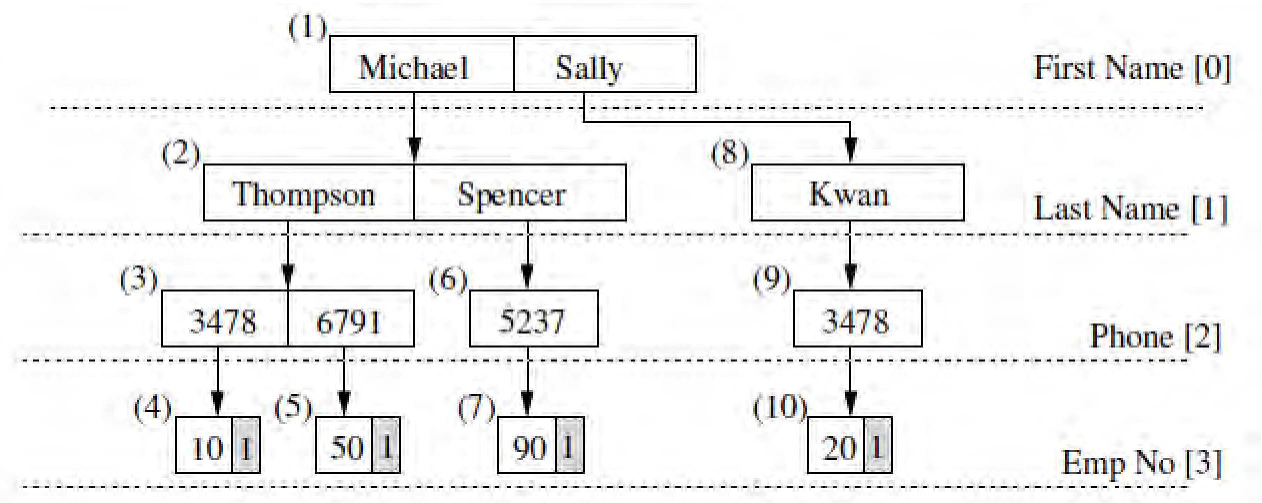
Gordian

- Yannis Sismanis et al. "GORDIAN: efficient and scalable discovery of composite keys." *Proceedings of the international conference on Very Large Data Bases*. 2006.
- Row-based algorithm

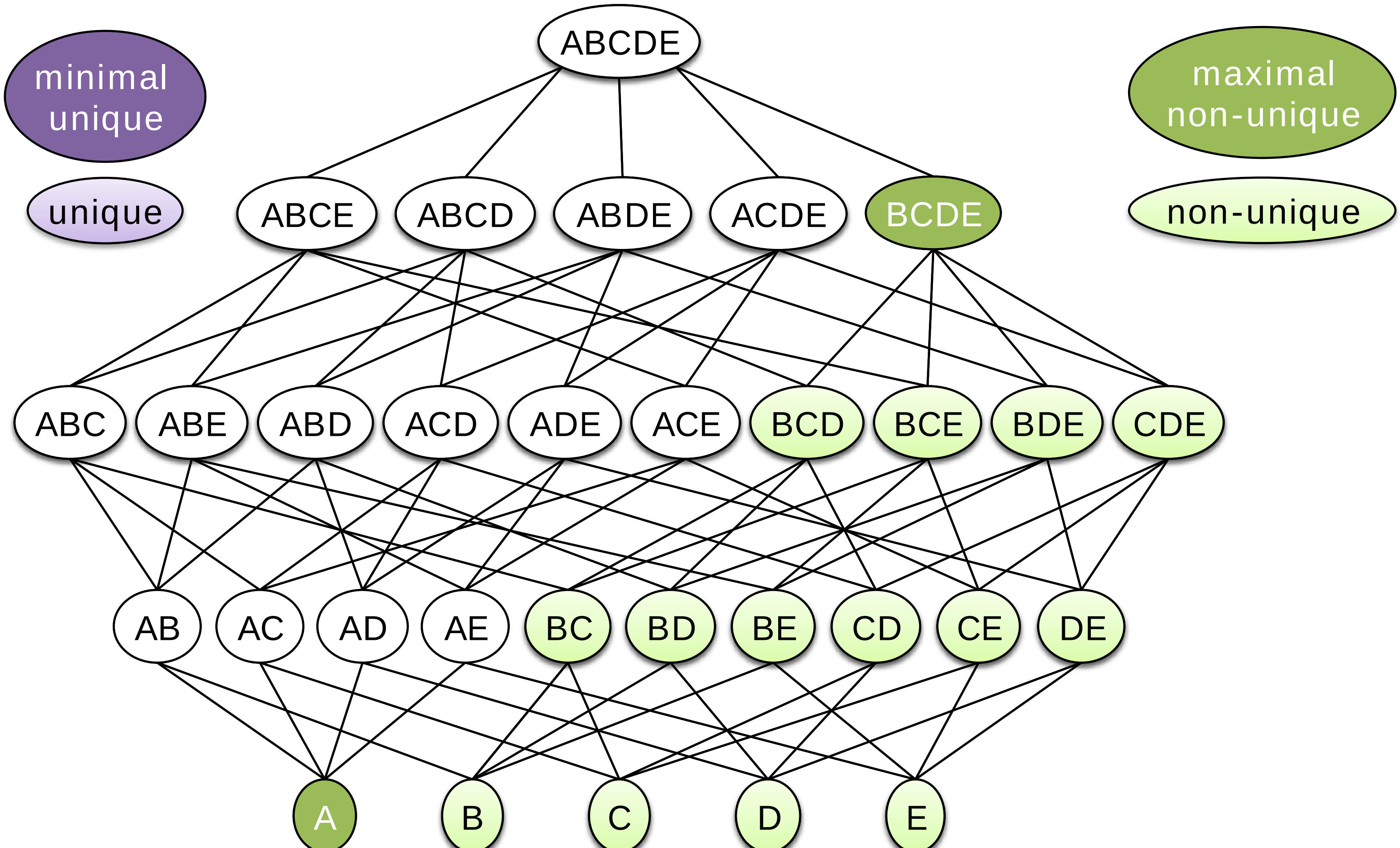
- Builds prefix tree while reading data
- Determines maximal non-uniques
- Compute minimal uniques from maximal non-uniques

Prefix tree

FirstName	*LastName*	*Phone*	*EmpNo*	*COUNT*
Michael	Thompson	3478	10	1
Sally	Kwan	3478	20	1
Michael	Spencer	5237	90	1
Michael	Thompson	6791	50	1



Calculating minimal uniques



Analysis Gordian

- According to paper, polynomial in the number of tuples for data with a Zipfian distribution of values
  - Can abort scan as soon as duplicate has been found

- Worst case
  - Exponential in the number of columns
  - All data needs to be stored in memory

- Computing minimal uniques from maximal non-uniques
  - $O(uniques^2 \cdot columns)$
  - Can be sped up with presorted list

Outlook

- Finding primary keys
  - Uniqueness is necessary criteria
  - No null values
  - Include other features
    - Name includes “id”, number of columns
- Approximate uniques
  - 99.9% of the data unique
  - Useful to detect data errors
  - Gordian, HCA, and DUCC can be easily modified
- Heuristics with sampling