Scalable Algorithms for Association Mining

Ivar Blohm, Erik Ström, Oskar Åsbrink

Lattice Prerequisites — Important Concepts:

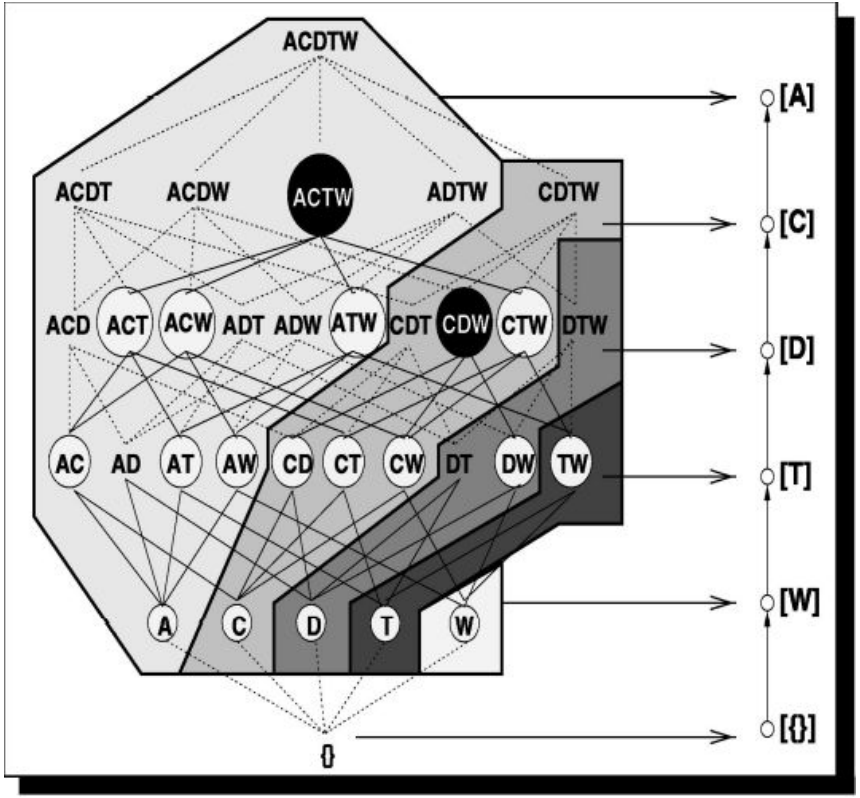
- Partial order
- X [Y (x is covered by y)
- Upper/Lower bound
- Top/Bottom elements
- Atoms

Prefix-Based (Equivalence) Classes

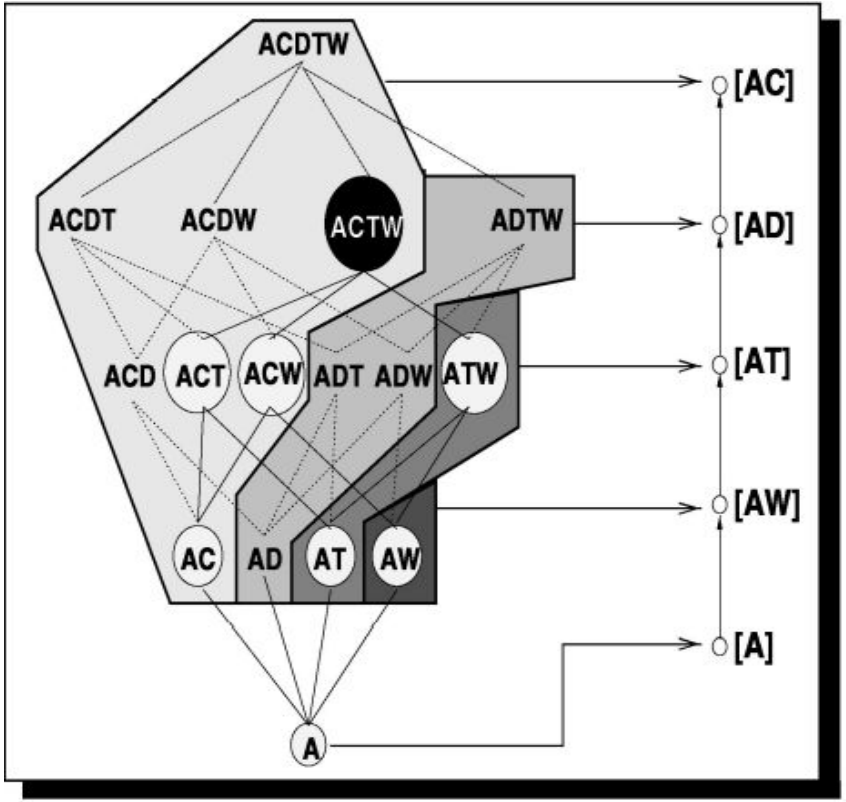
- p(X, k) = X[1 : k], the *k* length prefix of X
- θ_k := equivalence relation defined by the *k*-length prefix function
- Two itemsets are considered equivalent if they have the same k-length prefix
- An equivalence class is the set of itemsets that are equivalent under θ_k

Prefix-Based (Equivalence) Classes

(a) Equivalence classes induced by θ_1



(b) Equivalence classes induced by θ_2 on [A]



Support Counting

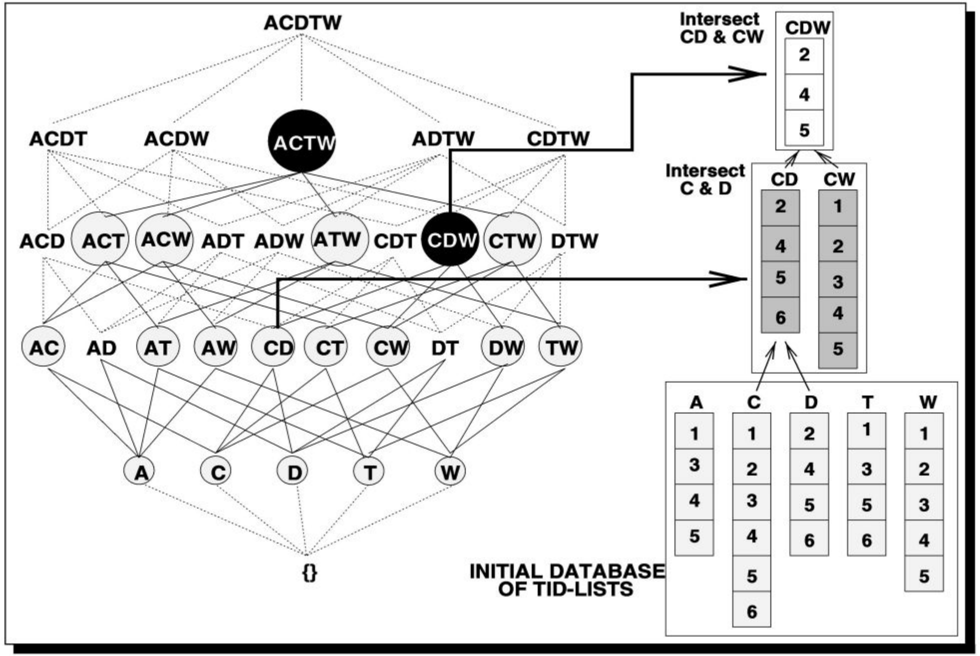
- $\mathcal{L}$(X) denotes, where X is an atom, the *tid-list* of X.
- We can calculate the support of (A ∩ B) by considering $\mathcal{L}$(A) ∩ $\mathcal{L}$(B)

tid-list for atoms A, C, …, W

A	C	D	T	W
1	1	2	1	1
3	2	4	3	2
4	3	5	5	3
5	4	6	6	4
	5			5
	6			

Support Counting

$$\sigma(A \cap B) = \frac{|\mathcal{L}(A) \cap \mathcal{L}(B)|}{\text{(total transactions)}}$$

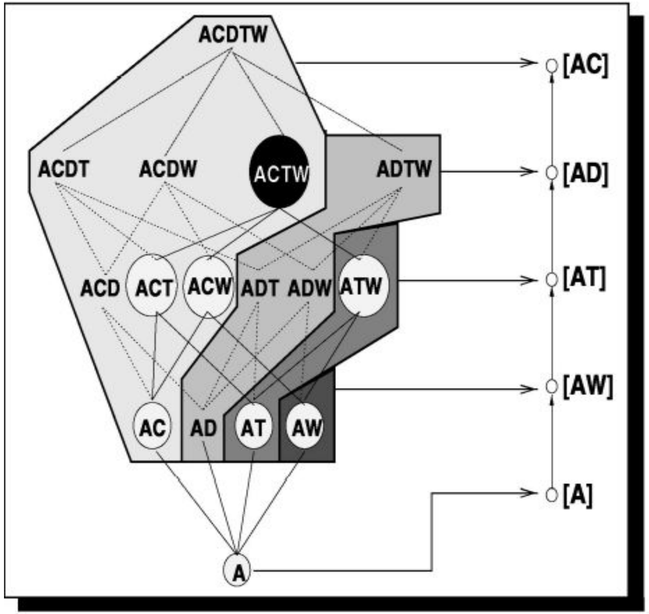


Bottom-up Search

Bottom-Up(S):
for all atoms $A_i \in S$ **do**
  $T_i = \emptyset$;
  for all atoms $A_j \in S$, with $j > i$ **do**
    $R = A_i \cup A_j$;
    $\mathcal{L}(R) = \mathcal{L}(A_i) \cap \mathcal{L}(A_j)$;
    if $\sigma(R) \geq min\_sup$ **then**
      $T_i = T_i \cup \{R\}$; $\mathcal{F}_{|R|} = \mathcal{F}_{|R|} \cup \{R\}$;
  end
end
for all $T_i \neq \emptyset$ **do** Bottom-Up(T_i);

ECLAT uses prefixed based equivalence relation θ_1 along with bottom-up search.

The recursive step



Properties of ECLAT

- Horizontal vs vertical tid-list
- Requires only a few database scans
- Recursive step in algorithm decompose the lattice into sublattices

Transaction No	Products
1	beer, wine, cheese
2	beer, potato chips
3	eggs, flour, butter, cheese
4	eggs, flour, butter, beer, potato chips
5	wine, cheese
6	potato chips
7	eggs, flour, butter, wine, cheese
8	eggs, flour, butter, beer, potato chips
9	wine, beer
10	beer, potato chips
11	butter, eggs
12	beer, potato chips
13	flour, eggs
14	beer, potato chips
15	eggs, flour, butter, wine, cheese
16	beer, wine, potato chips, cheese
17	wine, cheese
18	beer, potato chips
19	wine, cheese
20	beer, potato chips

(a) Horizontal data structure

Product	Transaction ID set
Wine	1, 5, 7, 9, 15, 16, 17, 19
Cheese	1, 3, 5, 7, 15, 16, 17, 19
Beer	1, 2, 4, 8, 9, 10, 12, 14, 16, 18, 20
Potato Chips	2, 4, 6, 8, 10, 12, 14, 16, 18, 20
Eggs	3, 4, 7, 8, 11, 13, 15
Flour	3, 4, 7, 8, 13, 15
Butter	3, 4, 7, 8, 11, 15

(b) Vertical tid-list

Market Basket optimization data

2	burgers,meatballs,eggs
3	chutney
4	turkey,avocado
5	mineral water,milk,energy bar,whole wheat rice,green tea
6	low fat yogurt
7	whole wheat pasta,french fries
8	soup,light cream,shallot
9	frozen vegetables,spaghetti,green tea
10	french fries
11	eggs,pet food
12	cookies
13	turkey,burgers,mineral water,eggs,cooking oil
14	spaghetti,champagne,cookies
15	mineral water,salmon
16	mineral water
17	shrimp,chocolate,chicken,honey,oil,cooking oil,low fat yogurt
18	turkey,eggs

Instacart market basket data

	order_id	product_id	add_to_cart_order	reordered
0	1	49302	1	1
1	1	11109	2	1
2	1	10246	3	0
3	1	49683	4	0
4	1	43633	5	1
5	1	13176	6	0
6	1	47209	7	0
7	1	22035	8	1

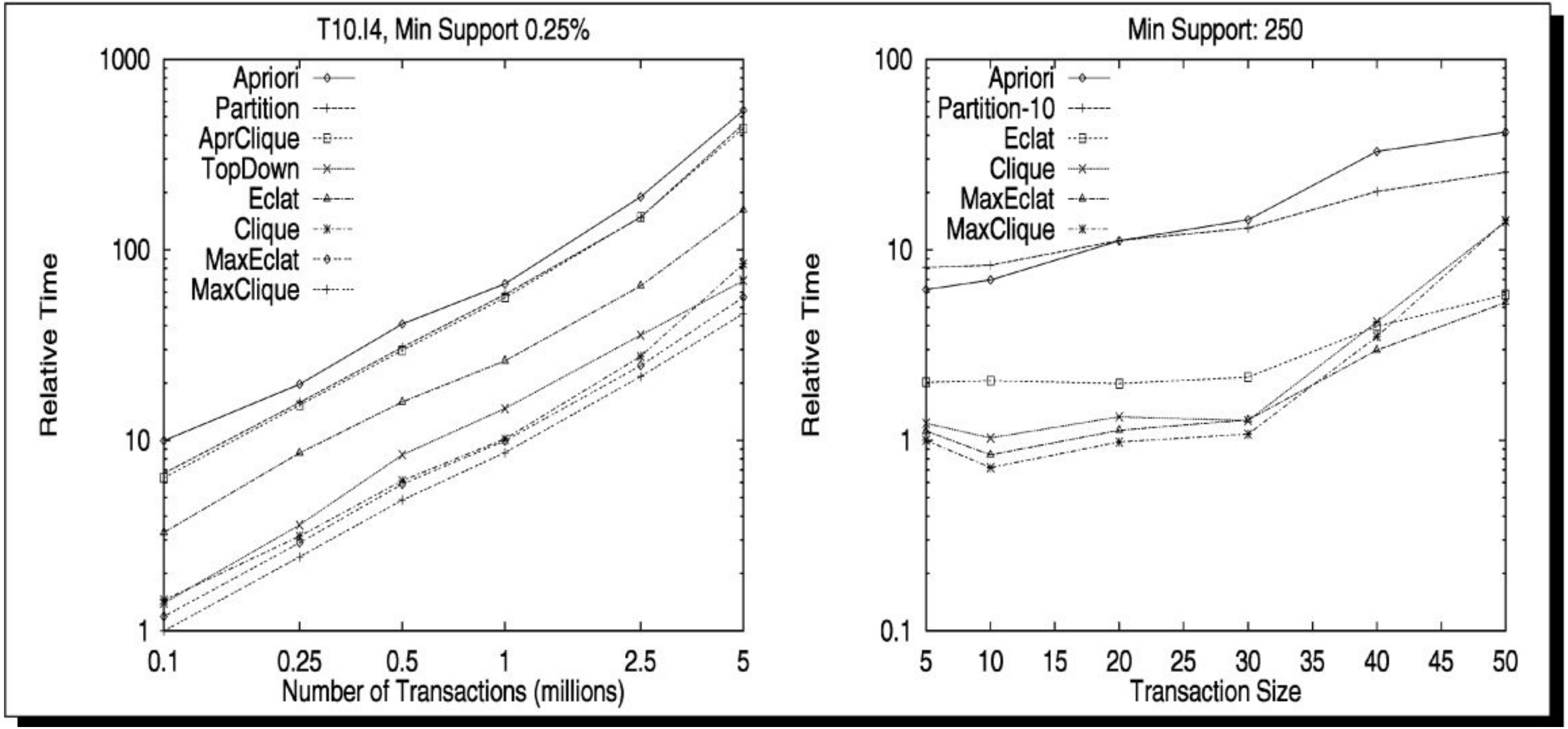
Algorithm performance across different libraries, 7501 transactions, support=0.05:

- Naïve python eclat (frequent itemsets only): 3.42s
- PyECLAT: Wall time: 16.2 s
- pyfpgrowth: Wall time: 49.5 s
- apyori: <0.0005s

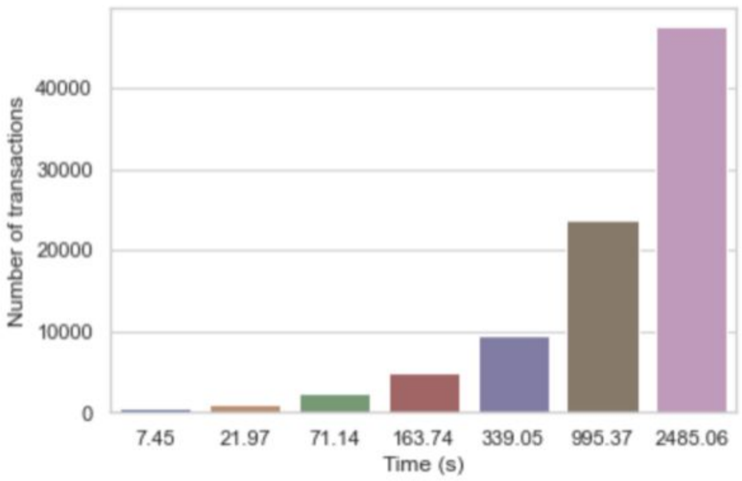
Why?

➔ Different levels of optimization
➔ Libraries calculate more or less things
➔ Python is slow

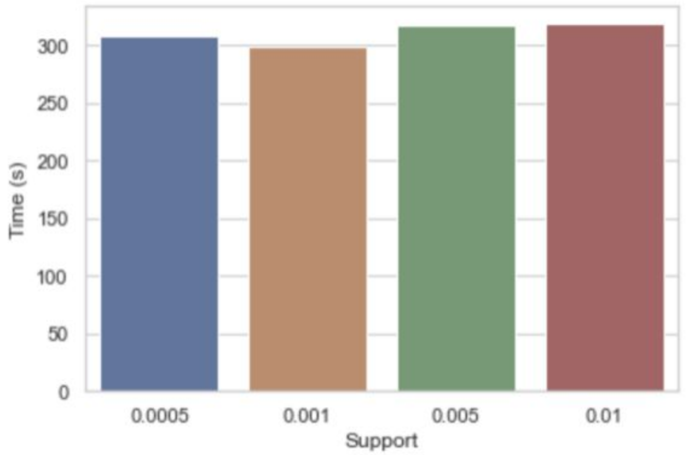
Results from the article



Execution time - ECLAT implementation



(a) Time of executions given different number of transactions



(b) Time of executions given different support

Performance considerations

- If the *tid-list* is stored in binary format, calculating support amounts to applying a boolean *and* operation on the respective lists!
- Utilize the fact that eclat can be applied to arbitrary equivalence classes under θ_k
  - We can choose k so that our itemsets fit in memory!
  - We are not restricted in *how* we compute the initial F_k we use as input!

  For example:

  - Recursion overhead is too expensive?
    - Precompute F_k for some k
  - Converting database from horizontal layout is too expensive?
    - Precompute F_k for some k

Conclusions

- State-of-the-art algorithm and optimized libraries
- Pros and cons with ECLAT
- What could we have done differently?

References

1. Scalable Algorithms for Association Mining (-> ECLAT) (https://dl.acm.org/doi/10.1109/69.846291)
2. www.kaggle.com/code/annadurbanova/market-basket-optimization-eclat-prac/data
3. www.kaggle.com/c/instacart-market-basket-analysis/data
4. https://towardsdatascience.com/the-eclat-algorithm-8ae3276d2d17 (examples (a), (b) on slide 8)