Comparison of Frequent Pattern Mining Algorithms

Archit Sengupta

October, 2024

1 Algorithms Tested

2 algorithms were tested for itemset mining

- Christian Borgelt's implementation of Apriori [1]
- Christian Borgelt's implementation of Eclat [1]

Christian Borgelt's implementations of Apriori and Eclat are optimized for frequent itemset mining. These algorithms are commonly used for tasks like market basket analysis. Each of them are suited for different use cases based on data layout and dataset size as we will show below.

1.1 Apriori Algorithm

Apriori is a breadth-first search algorithm that generates candidate itemsets and checks their frequencies in the dataset. It employs the **downward closure property (also known as the Apriori property)**, which states that if an itemset is frequent, all its subsets must also be frequent.

- **Prefix Tree Representation**: Borgelt's Apriori implementation uses a prefix tree to represent candidate itemsets. This helps in organizing itemsets based on the shared prefixes and leads to better processing and pruning of itemsets which are less frequent.[1]
- **Recursive Counting**: Uses a doubly recursive counting scheme. The transactions are processed in a recursive manner to update the counts of relevant itemsets. So for large transactions the computation overhead and complexity is lower..[1]

While improved from the original, it still requires multiple scans of the dataset. This makes it slower for large datasets as we will see in the tests conducted further.

Pros: Simple, well-optimized for smaller datasets, flexible parameters. **Cons**: Multiple scans, inefficient for large datasets.

1.2 Eclat Algorithm

Eclat (*Equivalence Class Clustering and bottom-up Lattice Traversal*) is a depth-first search algorithm that uses a **vertical data format** (transaction ID list, or TID) to represent the transactions. Each itemset is associated with a list of transaction IDs in which it appears. Instead of generating candidate itemsets like Apriori, Eclat intersects the TID lists of itemsets to find frequent itemsets.

This approach makes Eclat faster for large, sparse datasets, but it uses more memory because it stores the transaction ID lists explicitly. Eclat needs more structures in order to store the itemsets which have been found, when it is filtering for closed or maximal sets. This is because the earlier itemsets are forgotten and it backtracks during the depth first search.

Sparse Bit Matrix: Borgelt's Eclat represents transactions as a sparse bit matrix. The rows correspond to items and columns to transactions. This improves the algorithm because of better intersections of itemsets by manipulating bit matrices, reducing the need for extensive candidate generation.[1]

The algorithm switches between dense and sparse bit matrix representations depending on how dense the data is. This improves memory consumption

Pros: Efficient for large datasets, less number of database scans. **Cons**: Very high memory usage.

2 Datasets

Two different Datasets were used:

- **Chess Dataset** prepared by Roberto Bayardo from the UCI datasets and PUMSB. Smaller, with fewer transactions and less complexity. The itemsets tend to be smaller and less dense. http://fimi.uantwerpen.be/data/chess.dat.gz [2]
- **Accidents** Dataset by Karolien Geurts and contains (anonymized) traffic accident data. It is larger and denser, with more transactions and higher complexity. The number of frequent itemsets is much higher and so it leads to more computational challenges. http://fimi.uantwerpen.be/data/accidents.dat [3]

3 Implementation

3.1 Maintaining Support Threshold at 10%

2 different implementations were made using each algorithm on each dataset :

- Max subset size of 5
- Max subset size of 6

(Smaller max subset sizes were picked due to resource constraints as time taken increases exponentially as we try to combine more items)

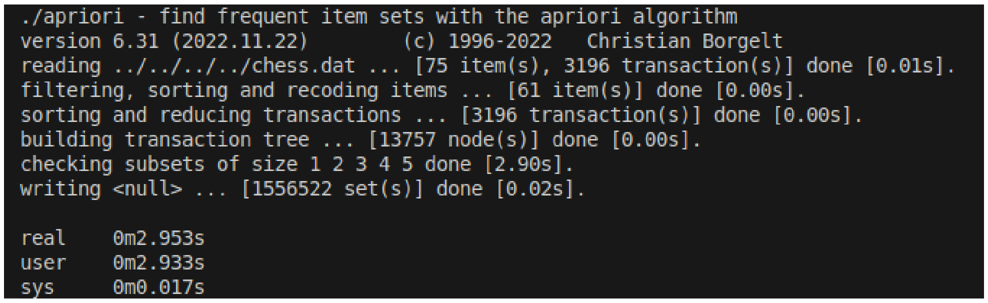
In all cases the support threshold for an itemset/rule is 10% .

3.1.1 Chess Dataset:

Apriori

- *Max item subset size: 5*

  Time taken: 2.9s

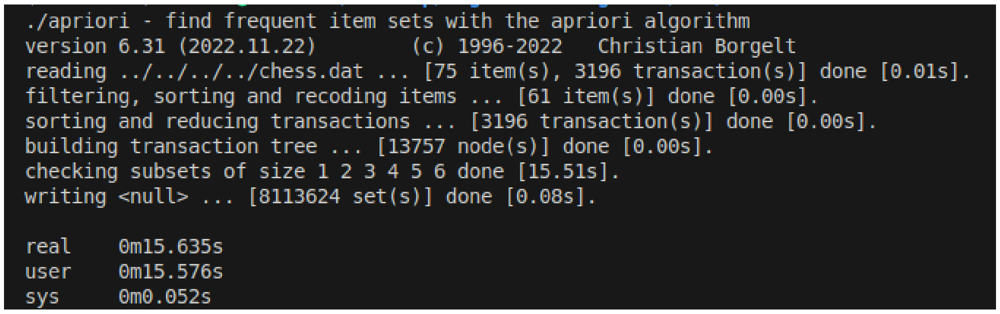


```
./apriori - find frequent item sets with the apriori algorithm
version 6.31 (2022.11.22)        (c) 1996-2022   Christian Borgelt
reading ../../../../chess.dat ... [75 item(s), 3196 transaction(s)] done [0.01s].
filtering, sorting and recoding items ... [61 item(s)] done [0.00s].
sorting and reducing transactions ... [3196 transaction(s)] done [0.00s].
building transaction tree ... [13757 node(s)] done [0.00s].
checking subsets of size 1 2 3 4 5 done [2.90s].
writing <null> ... [1556522 set(s)] done [0.02s].

real    0m2.953s
user    0m2.933s
sys     0m0.017s
```

- *Max item subset size: 6*

  Time taken: 15.6s



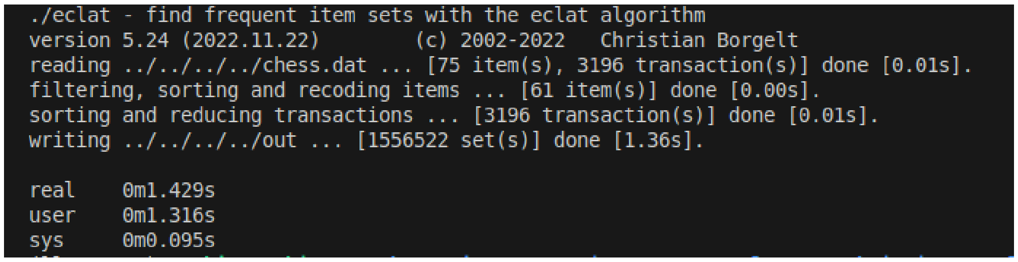
```
./apriori - find frequent item sets with the apriori algorithm
version 6.31 (2022.11.22)        (c) 1996-2022   Christian Borgelt
reading ../../../../chess.dat ... [75 item(s), 3196 transaction(s)] done [0.01s].
filtering, sorting and recoding items ... [61 item(s)] done [0.00s].
sorting and reducing transactions ... [3196 transaction(s)] done [0.00s].
building transaction tree ... [13757 node(s)] done [0.00s].
checking subsets of size 1 2 3 4 5 6 done [15.51s].
writing <null> ... [8113624 set(s)] done [0.08s].

real    0m15.635s
user    0m15.576s
sys     0m0.052s
```

Eclat

- *Max item subset size: 5*

  Time taken: 1.43s

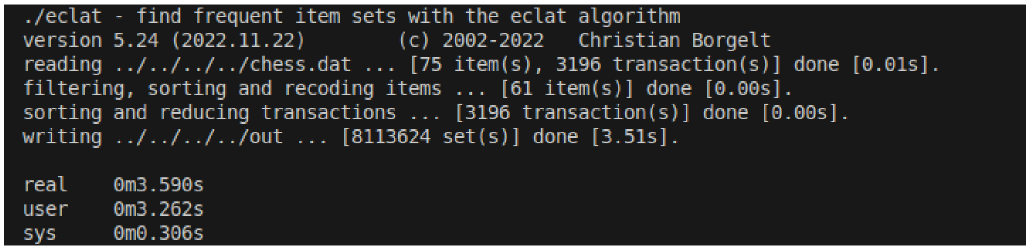


```
./eclat - find frequent item sets with the eclat algorithm
version 5.24 (2022.11.22)        (c) 2002-2022   Christian Borgelt
reading ../../../../chess.dat ... [75 item(s), 3196 transaction(s)] done [0.01s].
filtering, sorting and recoding items ... [61 item(s)] done [0.00s].
sorting and reducing transactions ... [3196 transaction(s)] done [0.01s].
writing ../../../../out ... [1556522 set(s)] done [1.36s].

real    0m1.429s
user    0m1.316s
sys     0m0.095s
```

- *Max item subset size: 6*

  Time taken: 3.6s



```
./eclat - find frequent item sets with the eclat algorithm
version 5.24 (2022.11.22)        (c) 2002-2022   Christian Borgelt
reading ../../../../chess.dat ... [75 item(s), 3196 transaction(s)] done [0.01s].
filtering, sorting and recoding items ... [61 item(s)] done [0.00s].
sorting and reducing transactions ... [3196 transaction(s)] done [0.00s].
writing ../../../../out ... [8113624 set(s)] done [3.51s].

real    0m3.590s
user    0m3.262s
sys     0m0.306s
```

Analysis:

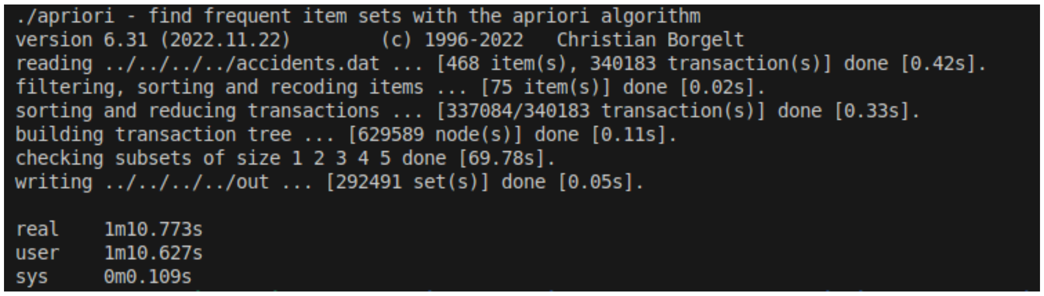
- **Apriori** performs moderately on the Chess dataset but shows a big increase in runtime (from 2.9s to 15.6s) as the subset size grows to include more items. This is because of Apriori's **breadth-first search**, which generates and tests many candidate itemsets. This becomes more and more as the itemset size increases. Each level of the search requires scanning the dataset and the number of candidates increases exponentially.
- **Eclat**, on the other hand, exhibits much lower runtimes (1.43s to 3.6s). There is only a small increase as the itemset size grows. Eclat's **depth-first search** and **TID list intersections** allow it to prune the search space in a better way. It does not face the candidate generation overhead that Apriori has. Eclat is more efficient even as the subset size increases.

3.1.2 Accidents Dataset:

Apriori

- *Max item subset size: 5*

  Time taken: 70.7s

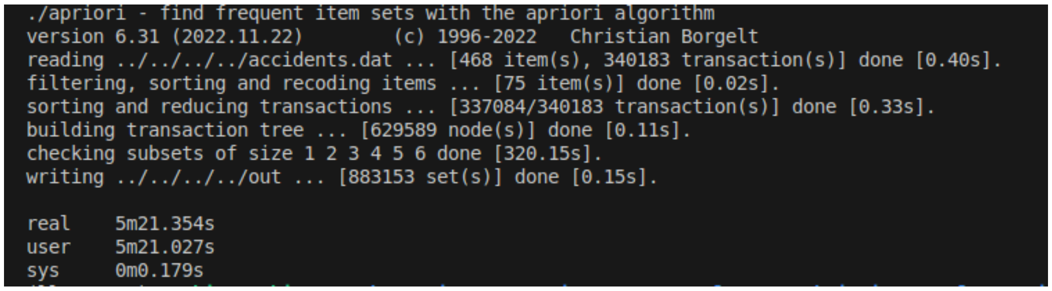


```
./apriori - find frequent item sets with the apriori algorithm
version 6.31 (2022.11.22)        (c) 1996-2022   Christian Borgelt
reading ../../../../accidents.dat ... [468 item(s), 340183 transaction(s)] done [0.42s].
filtering, sorting and recoding items ... [75 item(s)] done [0.02s].
sorting and reducing transactions ... [337084/340183 transaction(s)] done [0.33s].
building transaction tree ... [629589 node(s)] done [0.11s].
checking subsets of size 1 2 3 4 5 done [69.78s].
writing ../../../../out ... [292491 set(s)] done [0.05s].

real    1m10.773s
user    1m10.627s
sys     0m0.109s
```

- *Max item subset size: 6*

  Time taken: 321.4s



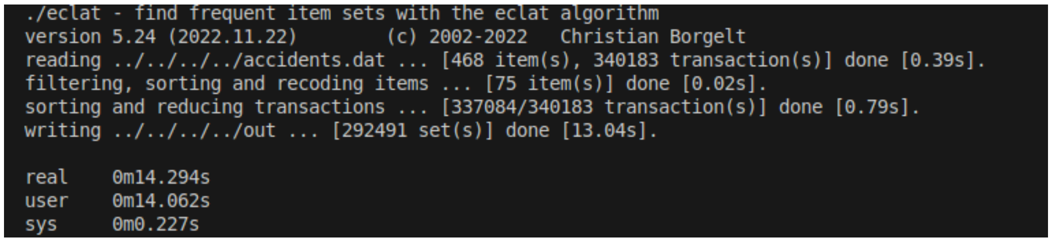
```
./apriori - find frequent item sets with the apriori algorithm
version 6.31 (2022.11.22)        (c) 1996-2022   Christian Borgelt
reading ../../../../accidents.dat ... [468 item(s), 340183 transaction(s)] done [0.40s].
filtering, sorting and recoding items ... [75 item(s)] done [0.02s].
sorting and reducing transactions ... [337084/340183 transaction(s)] done [0.33s].
building transaction tree ... [629589 node(s)] done [0.11s].
checking subsets of size 1 2 3 4 5 6 done [320.15s].
writing ../../../../out ... [883153 set(s)] done [0.15s].

real    5m21.354s
user    5m21.027s
sys     0m0.179s
```

Eclat

- *Max item subset size: 5*

  Time taken: 14.3s

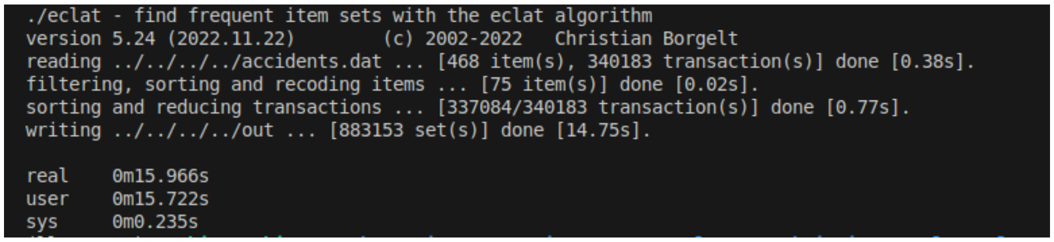


```
./eclat - find frequent item sets with the eclat algorithm
version 5.24 (2022.11.22)        (c) 2002-2022   Christian Borgelt
reading ../../../../accidents.dat ... [468 item(s), 340183 transaction(s)] done [0.39s].
filtering, sorting and recoding items ... [75 item(s)] done [0.02s].
sorting and reducing transactions ... [337084/340183 transaction(s)] done [0.79s].
writing ../../../../out ... [292491 set(s)] done [13.04s].

real    0m14.294s
user    0m14.062s
sys     0m0.227s
```

- *Max item subset size: 6*

  Time taken: 16.0s



```
./eclat - find frequent item sets with the eclat algorithm
version 5.24 (2022.11.22)        (c) 2002-2022   Christian Borgelt
reading ../../../../accidents.dat ... [468 item(s), 340183 transaction(s)] done [0.38s].
filtering, sorting and recoding items ... [75 item(s)] done [0.02s].
sorting and reducing transactions ... [337084/340183 transaction(s)] done [0.77s].
writing ../../../../out ... [883153 set(s)] done [14.75s].

real    0m15.966s
user    0m15.722s
sys     0m0.235s
```

Analysis:

- **Apriori's performance degrades drastically** on the Accidents dataset, with runtimes increasing significantly to 70.7s and 321.4s as the max itemset size grows from 5 to 6. This is because the Accidents dataset is much larger and denser. It leads to an overwhelming number of candidate itemsets that Apriori needs to generate and test. The multiple database scans required for each level in Apriori causes poor scalability in large datasets.
- **Eclat handles the Accidents dataset far more efficiently**, with runtimes of 14.3s and 16.0s for max itemsets of sizes 5 and 6. The use of **vertical data structures (TID lists)** enables Eclat to intersect transactions quickly without generating a full list of candidates. The sparse bit matrix also helps reduce candidate generation. This results in much smaller increases in runtime even as the dataset grows in complexity.

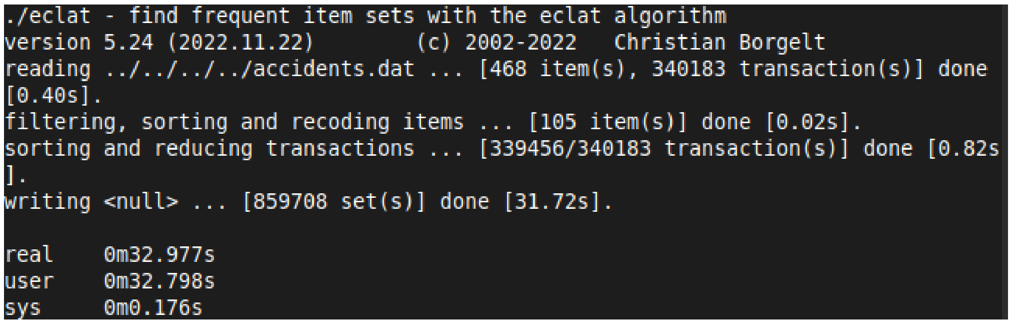
3.2 Reducing Minimum Support

In order to further test performance the support for an item or itemset was lowered, to incorporate more patterns for consideration. The Accidents dataset was used again with a max itemset size of 5.

The chosen values are 5% and 3%

Eclat:

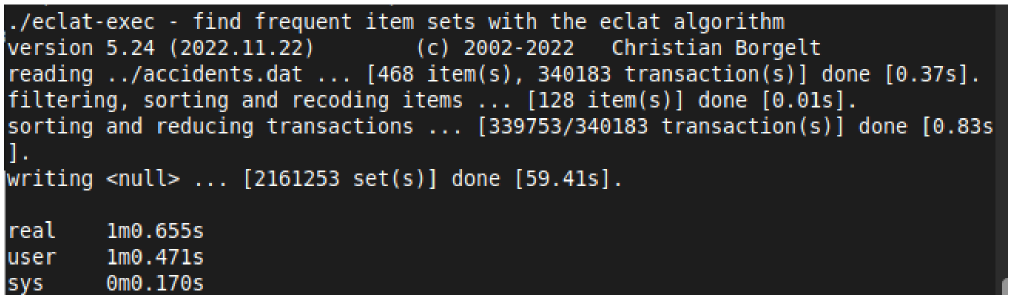
- *5%: 32.98s*



```
./eclat - find frequent item sets with the eclat algorithm
version 5.24 (2022.11.22)        (c) 2002-2022   Christian Borgelt
reading ../../../../accidents.dat ... [468 item(s), 340183 transaction(s)] done
[0.40s].
filtering, sorting and recoding items ... [105 item(s)] done [0.02s].
sorting and reducing transactions ... [339456/340183 transaction(s)] done [0.82s
].
writing <null> ... [859708 set(s)] done [31.72s].

real    0m32.977s
user    0m32.798s
sys     0m0.176s
```

- *2.5%: 60.6s*

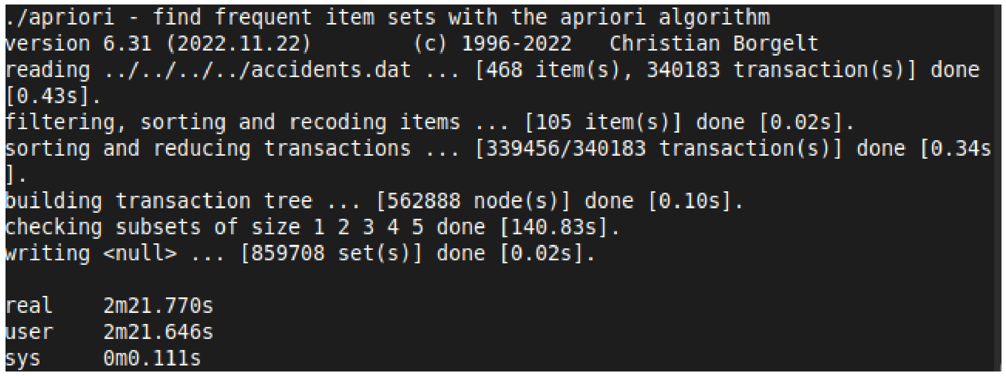


```
./eclat-exec - find frequent item sets with the eclat algorithm
version 5.24 (2022.11.22)        (c) 2002-2022   Christian Borgelt
reading ../accidents.dat ... [468 item(s), 340183 transaction(s)] done [0.37s].
filtering, sorting and recoding items ... [128 item(s)] done [0.01s].
sorting and reducing transactions ... [339753/340183 transaction(s)] done [0.83s
].
writing <null> ... [2161253 set(s)] done [59.41s].

real    1m0.655s
user    1m0.471s
sys     0m0.170s
```

Apriori:

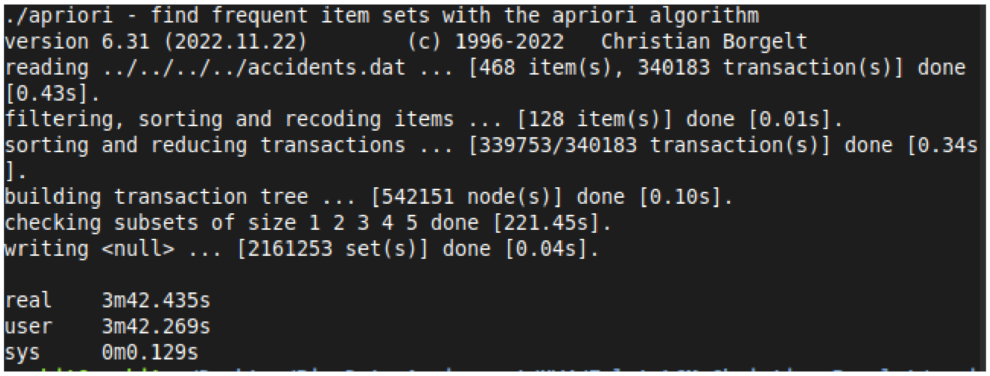
- *5% : 141.77s*



```
./apriori - find frequent item sets with the apriori algorithm
version 6.31 (2022.11.22)        (c) 1996-2022   Christian Borgelt
reading ../../../../accidents.dat ... [468 item(s), 340183 transaction(s)] done
[0.43s].
filtering, sorting and recoding items ... [105 item(s)] done [0.02s].
sorting and reducing transactions ... [339456/340183 transaction(s)] done [0.34s
].
building transaction tree ... [562888 node(s)] done [0.10s].
checking subsets of size 1 2 3 4 5 done [140.83s].
writing <null> ... [859708 set(s)] done [0.02s].

real    2m21.770s
user    2m21.646s
sys     0m0.111s
```

- *2.5%: 222.44s*



```
./apriori - find frequent item sets with the apriori algorithm
version 6.31 (2022.11.22)        (c) 1996-2022   Christian Borgelt
reading ../../../../accidents.dat ... [468 item(s), 340183 transaction(s)] done
[0.43s].
filtering, sorting and recoding items ... [128 item(s)] done [0.01s].
sorting and reducing transactions ... [339753/340183 transaction(s)] done [0.34s
].
building transaction tree ... [542151 node(s)] done [0.10s].
checking subsets of size 1 2 3 4 5 done [221.45s].
writing <null> ... [2161253 set(s)] done [0.04s].

real    3m42.435s
user    3m42.269s
sys     0m0.129s
```

Analysis:

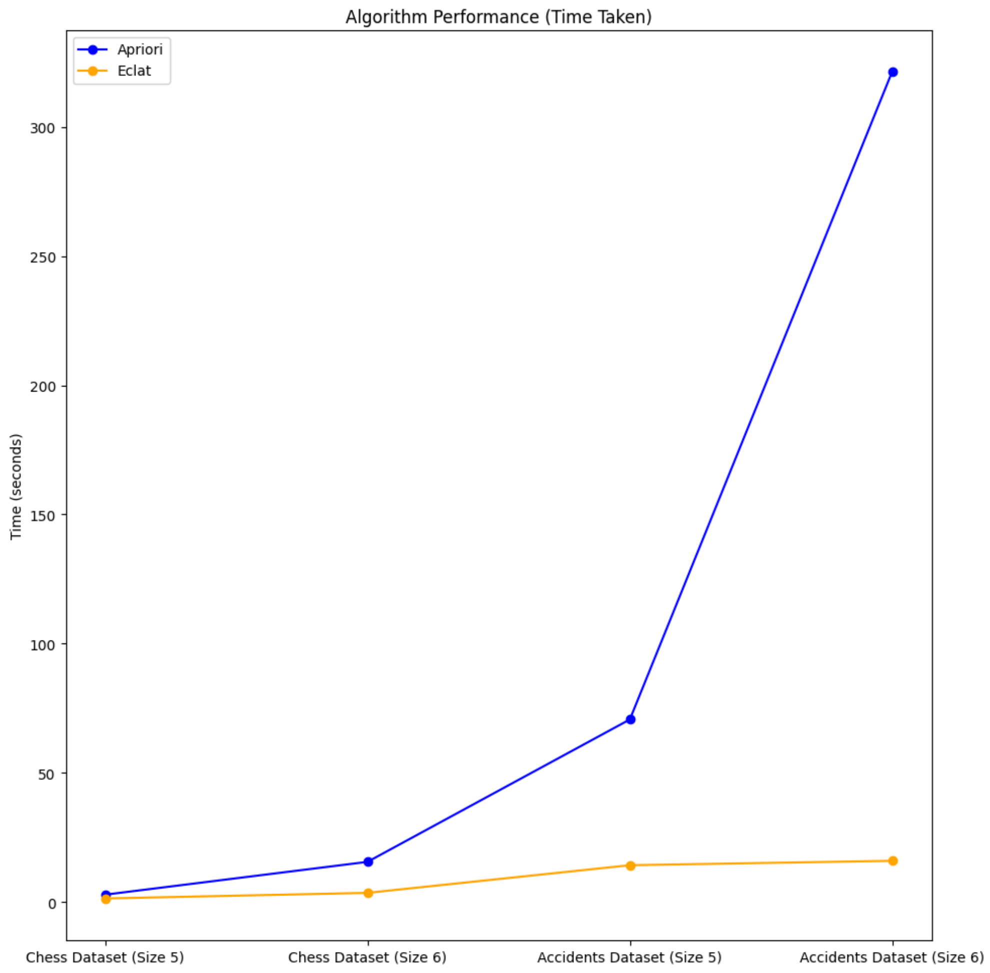
Both Apriori and Eclat show very similar scaling as the support threshold decreases, which is interesting. Lowering the threshold to 5% causes a ~2x increase in time for both. And decreasing it further to 2.5% reduced it further. Both algorithms seem to follow a roughly proportional increase in runtime as support is lowered.

This shows that while Apriori's overhead in generating candidates grows, it doesn't dominate the process as much as expected due to the optimizations made by Borgelt. Similarly, Eclat scales linearly, benefiting from efficient pruning of itemsets. While Eclat is obviously faster in all cases, they both handle lower support levels in a similar manner.

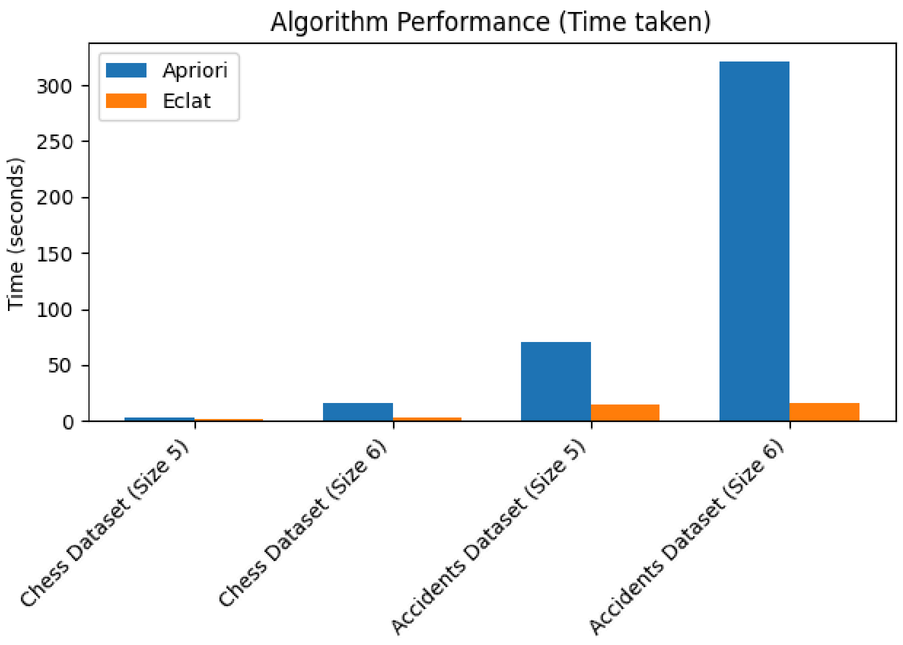
4 Results and Insights

4.1 Dataset and itemset comparison

Algorithm,Dataset	**Chess (Itemset Size 5)**	**Chess (Itemset Size 6)**	**Accidents (Itemset Size 5)**	**Accidents (Itemset Size 6)**
Apriori	2.9s	15.6s	70.7s	321.4s
Eclat	1.43s	3.6s	14.3s	16.0s



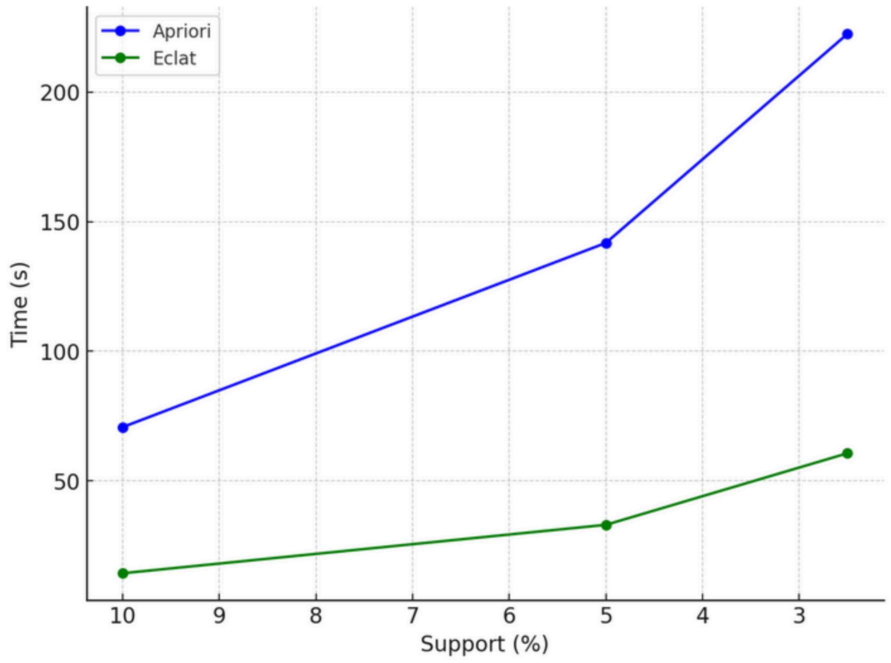
Algorithm Performance (Time Taken)
Apriori
Eclat
Time (seconds)
0
50
100
150
200
250
300
Chess Dataset (Size 5)
Chess Dataset (Size 6)
Accidents Dataset (Size 5)
Accidents Dataset (Size 6)



Algorithm Performance (Time taken)
Apriori
Eclat
Time (seconds)
0
50
100
150
200
250
300
Chess Dataset (Size 5)
Chess Dataset (Size 6)
Accidents Dataset (Size 5)
Accidents Dataset (Size 6)

4.2 Support Threshold Comparison

	10%	**5%**	**2.5%**
Apriori	70.7s	141.77s	222.44s
Eclat	14.3s	32.98s	60.6s



Key Insights:

1. **Apriori's Performance**:
   - **Apriori is much slower.** As seen in all test cases Apriori, despite the optimizations, is a much poorer algorithm.
   - **Scalability Issues**: Apriori faces issues with larger datasets like Accidents due to its **breadth-first search** method and repeated database scans for each level of the itemset size. This leads to **exponential growth** in candidate generation, which causes slowdowns.
   - **Sensitive to size**: While Apriori performs relatively well on smaller datasets like Chess(although much slower than Eclat), its performance deteriorates rapidly as the dataset size and complexity increase.
   - As more and more candidates/ itemsets need to be generated the performance gets a lot worse.

2. **Eclat's Performance**:
   - **Efficient for Large Datasets**: Eclat performs much better on larger datasets proving that the **depth-first traversal** technique and use of **TID lists** makes a big difference. The minimal increase in runtime for the Accidents dataset demonstrates **superior scalability**.
   - **Better Scaling with Itemset Size**: Eclat's runtime increases very little as itemset size grows, whereas Apriori's time grows exponentially. This makes Eclat better suited for large, dense datasets with high-dimensional data.
   - Even for **lower support thresholds** Eclat shows much faster performance than Apriori but the time taken does scale predictably.

5 References

[1] Christian Borgelt, Efficient Implementations of Apriori and Eclat
https://ceur-ws.org/Vol-90/borgelt.pdf
[2] Dataset 1: http://fimi.uantwerpen.be/data/chess.dat
[3] Dataset 2: http://fimi.uantwerpen.be/data/accidents.dat
[4] Source Code: https://borgelt.net/eclat.html (Contains both Eclat as well as Apriori)