COMP-4250 Big Data Analytics and Database Design

Project I Report

Mining Frequent Datasets

Author: Keigo Katanaga

Student ID: 110068805

Submitted to: Dr. Majid Afshar

October 25, 2024

Introduction

The purpose of this report is to compare the runtimes of 4 algorithms: A-Priori, Park-Chen-Yu (PCY), Multistage PCY, and Multihash PCY. These algorithms find frequent itemsets from multiple baskets of items.

Experiment Details

For the experiment, we are given a text file with 88162 lines/baskets to search for frequent singles and pairs of items. Finding triples and so on are not included in the experiment. Each basket contains multiple integer items.

Our parameters are the dataset size and support threshold. The dataset size ranges from 1%, 5%, 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, and 100% of the total baskets, also called chunks. This allows us to see the effectiveness of each algorithm as the data grows. The second parameter, support threshold, ranges from 1%, 5%, and 10% of our total baskets; support of items below the threshold are labeled infrequent, while support of items above the threshold are frequent.

The code is written in Python. It reads data from the text input (baskets) and runs the 4 algorithms for each combination of dataset size + support threshold. The program outputs 3 line graphs (1 for each support threshold) which show the runtime (ms) of the algorithms based on the dataset size.

Each algorithm is re-run 3 times for each dataset size, and the minimum runtime of the 3 re-runs is shown on the line graphs. This is done to mitigate the effects of runtime spikes to the graph caused by the CPU scheduler.

The PCY algorithms use the first hash function: *(i x j) mod N*
Where N is the next prime after the total number of unique singletons in the current chunk.

Multistage & Multihash PCY uses the second hash function: *(i + j) mod N*. Note that Multihash PCY uses *(i x j) mod (next prime of N / 2)* and *(i + j) mod (next prime of N / 2)* since it's splitting the memory.

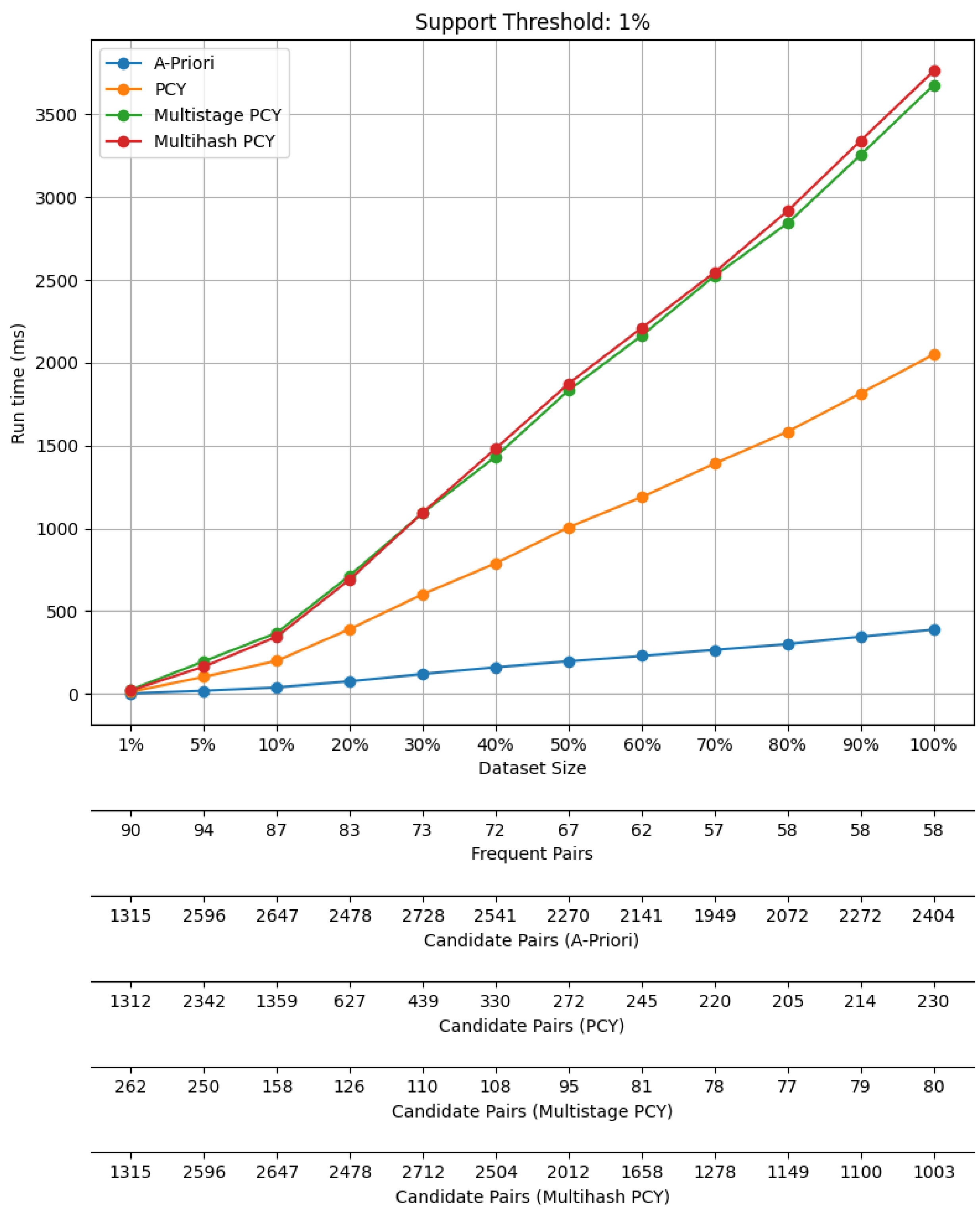
Below each graph is the number of frequent and candidate pairs found from each chunk, for each algorithm.

Specification

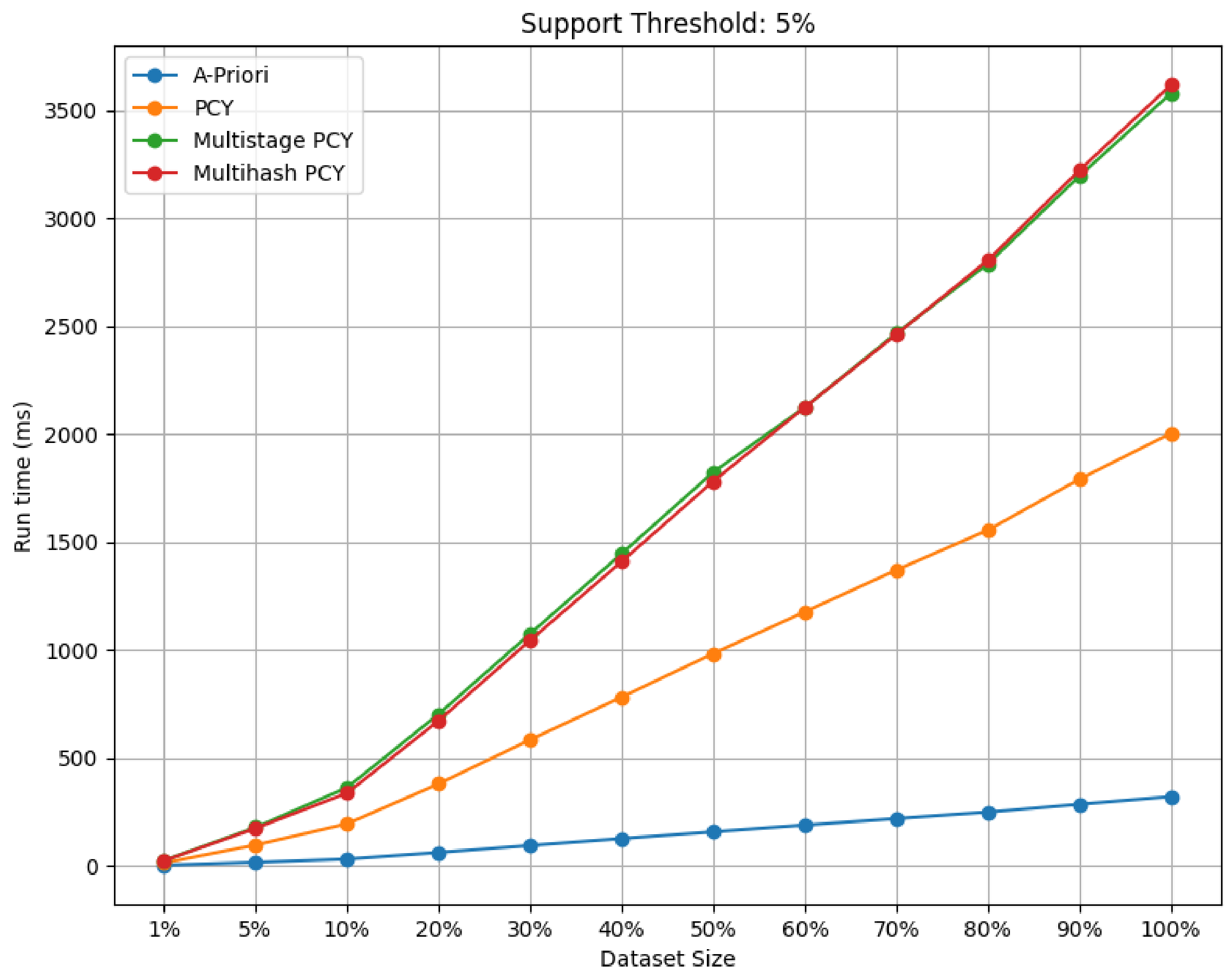
This experiment is done on the following system specs:

- CPU: AMD Ryzen 5 7600X
- GPU: AMD Radeon RX 6750 XT
- RAM: 32GB
- OS: Windows 10 Home Ver. 22H2

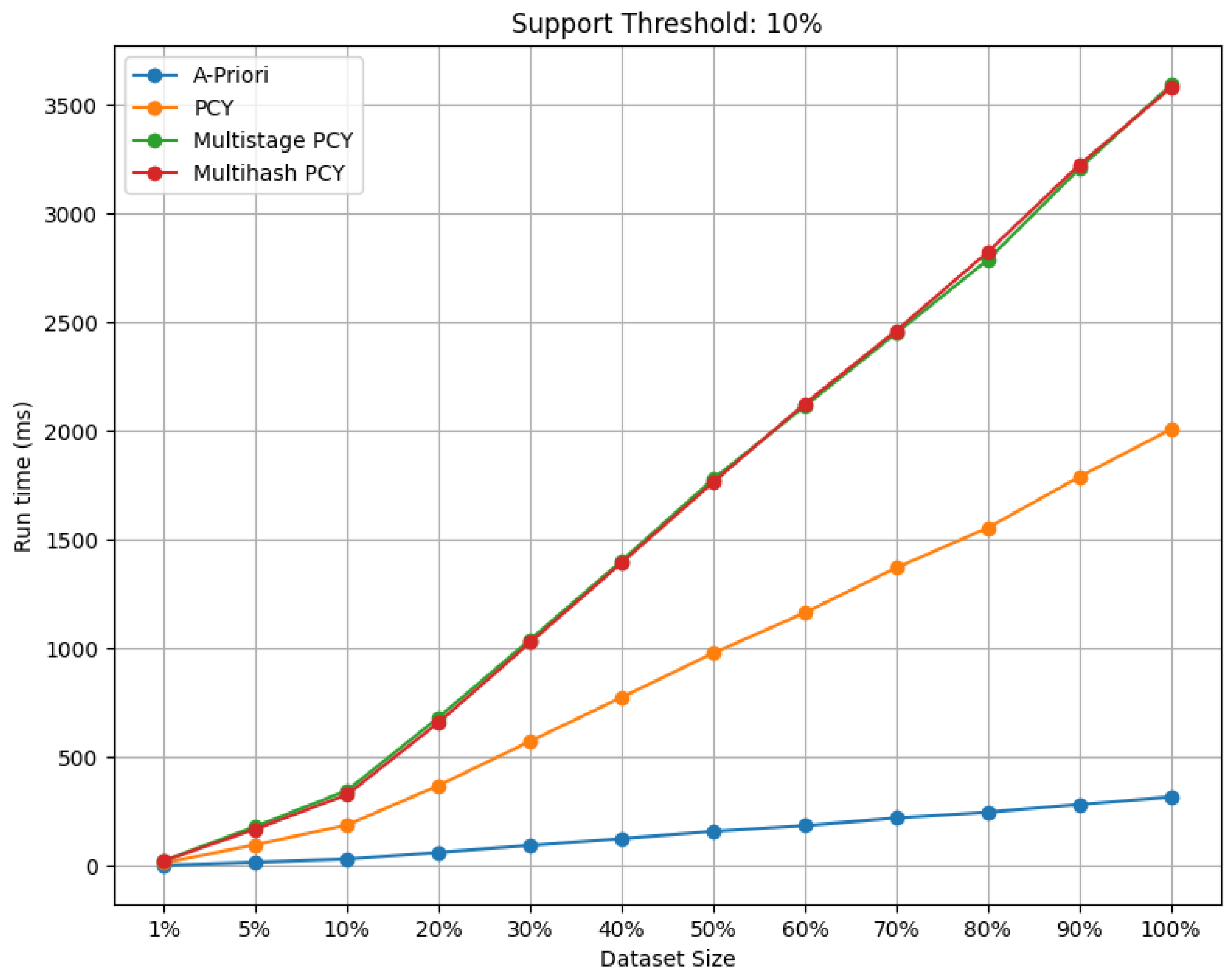
Results



Dataset Size	1%	5%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Frequent Pairs	90	94	87	83	73	72	67	62	57	58	58	58
Candidate Pairs (A-Priori)	1315	2596	2647	2478	2728	2541	2270	2141	1949	2072	2272	2404
Candidate Pairs (PCY)	1312	2342	1359	627	439	330	272	245	220	205	214	230
Candidate Pairs (Multistage PCY)	262	250	158	126	110	108	95	81	78	77	79	80
Candidate Pairs (Multihash PCY)	1315	2596	2647	2478	2712	2504	2012	1658	1278	1149	1100	1003



	1%	5%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Frequent Pairs	8	9	9	9	9	9	9	8	7	7	7	7
Candidate Pairs (A-Priori)	21	10	10	10	10	10	10	15	15	15	15	15
Candidate Pairs (PCY)	12	9	9	9	9	9	9	9	8	7	7	7
Candidate Pairs (Multistage PCY)	9	9	9	9	9	9	9	8	7	7	7	7
Candidate Pairs (Multihash PCY)	16	10	9	9	9	9	9	9	8	7	7	8



	1%	5%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Frequent Pairs	5	4	4	4	5	4	4	4	4	3	3	4
Candidate Pairs (A-Priori)	10	10	10	10	10	10	10	10	10	10	10	10
Candidate Pairs (PCY)	6	5	6	6	6	5	5	5	5	4	4	5
Candidate Pairs (Multistage PCY)	5	4	5	5	6	4	4	4	4	3	3	4
Candidate Pairs (Multihash PCY)	6	7	6	6	6	7	7	6	6	5	6	6

Based on the graphs, the A-Priori algorithm runs the fastest, even though PCY (and its other variants) is said to run faster than A-Priori. In addition, my A-Priori implementation computes not just the candidate pairs, but also frequent pairs. The other algorithms only compute for the candidate pairs!

This behavior is likely because PCY spends too much time generating the combinations of pairs for each basket on both passes, while A-Priori only generates the combinations on the second pass.

On the other hand, A-Priori generates the largest amount of candidate pairs out of the 4 algorithms. This makes sense as PCY and its variants are designed to save memory while reducing the number of candidate pairs. It is said that for PCY to beat A-Priori in terms of memory saved, PCY must eliminate 2/3 of candidate pairs generated by the 2nd pass of A-Priori. We can see that the number of candidate pairs of PCY and its variants are much lower than A-Priori as the dataset size goes up, especially with a lower support threshold.

The algorithm that selected candidate pairs closest to the actual frequent pairs is Multistage PCY. Its accuracy increased as the support threshold went up.

The worst algorithm was Multihash PCY. It ran as slow as Multistage while also being as inaccurate as A-Priori. This is likely due to the size of the hash map getting split into two to accommodate the use of two hash maps in one pass, thereby decreasing the effectiveness of the hash maps.

Conclusion

A-Priori and PCY algorithms perform with an inverse proportion of runtime to the memory usage and accuracy of each other.

A-Priori has the fastest runtime out of the 4 algorithms, thanks to its minimal computation. In exchange, a lot of memory is used for candidate pairs which are not even close in magnitude to the actual frequent pairs.

On the other hand, Multistage PCY has the slowest runtime (together with Multihash PCY) as it generates 2-combinations on all 3 passes as opposed A-Priori which generates it once. However, Multistage PCY selects the most accurate candidate pairs out of the 4 algorithms while also minimizing memory use by reducing the number of candidate pairs.

PCY and Multihash PCY sit in-between in terms of overall performance as PCY is faster than its variants but slower than A-Priori, and its accuracy is lower than Multihash PCY.