BigData: Apriori & SONs

A comparison among several implementations of Association-Rule based algorithms.

Cesare Montresor - VR481252

https://github.com/cesare-montresor/apriori-and-sons

Details

Project

The project have been developed as framework for benchmarking a series of frequent itemset algorithms against various datasets and minimum support levels.
All the experiments and results have been obtained by trying as much as possible to replicate equal testing conditions. Loading of data, running of the experiments, profiling and result storage are unified and are not included in the computation of the timings, just the pure execution of the algorithm.

Hardware

- Ubuntu 22.04
- 32GB RAM
- AMD Ryzen 9 5900X
- 12 Cores
- 24 Threads *(24 workers)*
- Spark 3.4.0 for Hadoop 3.3.4
- Python 3.10.9 (conda)

Algorithms

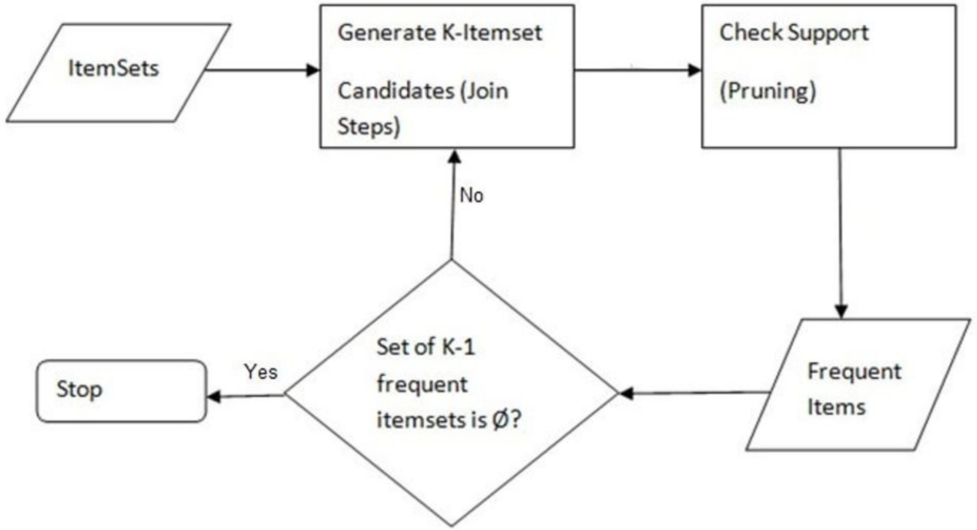
All implementations are based on PySpark and make use map, reduce, cartesian, filter and other typical paradigms from functional programming, replacing loops as much as possible.
Some implementations, like apriori_list are made in pure python as they are intended to be used on partition data, which by nature in Spark, is not an RDD, but simple python object. Also in this case the same guideline have been followed.

ALGORITHMS

Apriori

Steps

1. Count single transaction items
2. Obtain FrequentSet of size 1
3. Permutate to obtain candidates size 2
4. Perform cartesian between candidates and transactions.
5. Filter only existing sets for existing transactions and count them.
6. Obtain FrequentSet on size 2
7. Repeat from step 3


ItemSets
Generate K-Itemset Candidates (Join Steps)
Check Support (Pruning)
Frequent Items
Set of K-1 frequent itemsets is Ø?
No
Yes
Stop

Apriori: Focus

Extending the Set

```
extendedSets = product(itemSupport, itemUnique) ★
extendedSets = map(flatMergeCartesian, extendedSets)  # ( ('3','1'),'4') -> ('1','3','4')
extendedSets = tuple(filter(lambda item: len(item) == set_length, extendedSets))
# from flatMergeCartesian: ( ('4','3'), '3') -> ('3','4')
extendedSets = tuple(set(extendedSets))
```

Filtering the Set

```
existingSets = product(extendedSets, transactions) ★
existingSets = tuple(filter(lambda item: set(item[0]).issubset(item[1]), existingSets))
existingSets = tuple(map(lambda item: item[0], existingSets))
```

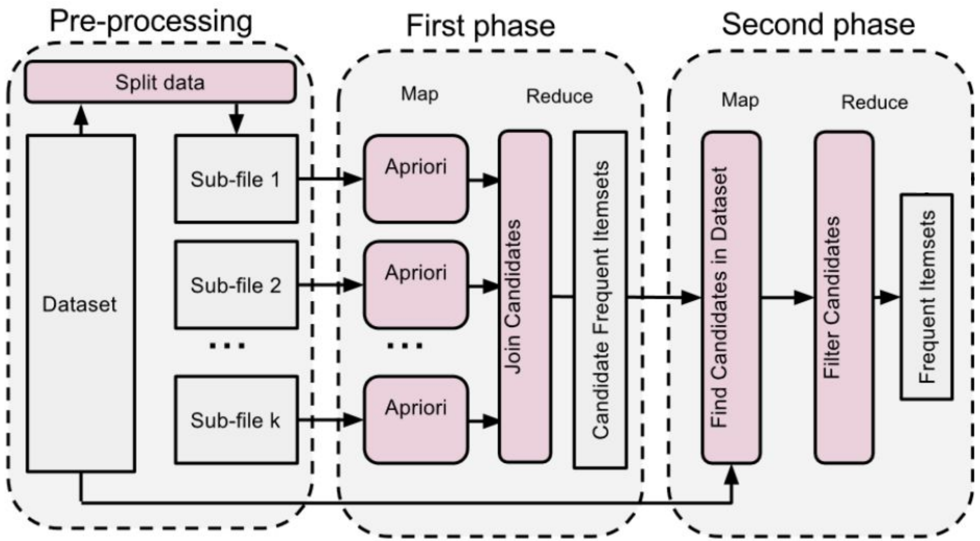
SON

Steps

1. Partition the dataset
2. Perform "apriori" for each partition, discard the counting.
3. Merge the results from all partitions and use those as candidates.
4. Perform cartesian between candidates and transactions.
5. Find candidate sets.
6. Filter candidates and obtain Frequent Itemsets.

1 (steps 2–4)

2 (steps 5–6)



SON: Focus

Find candidates from partitions

```
candidateSets = partitionSets.mapPartitions(lambda part: ★priori_list(list(part), min_support))
candidateSets = candidateSets.map(lambda item: item[0])
candidateSets = candidateSets.coalesce(1)
candidateSets = candidateSets.distinct()
```

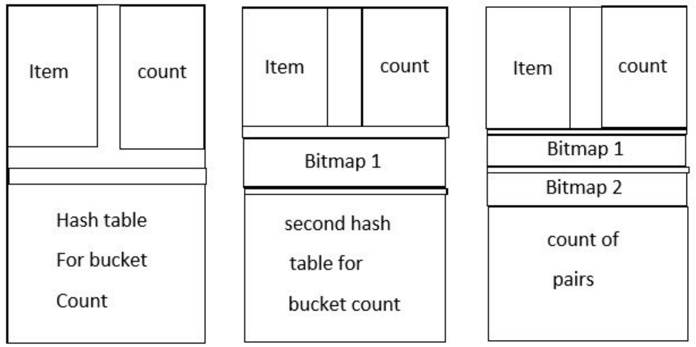
Test candidates

```
# test candidates
testingSets = transactions.cartesian(candidateSets) ★
testingSets = testingSets.filter(lambda item: set(item[0]).issubset(item[1]))
testingSets = testingSets.reduceByKey(lambda x, y: x + y)
testingSets = testingSets.filter(lambda item: item[1] > candidate_frequency)
```

PCY

Steps

1. Count single transaction items
2. Obtain FrequentSet of size 1
3. Partition the dataset into buckets
4. Generate Candidate set from row combination.
5. Using a hash function count items per bucket and extract candidates and use it for filtering.
6. Perform cartesian between candidates and transactions.
7. Find candidate sets.
8. Filter candidates and obtain Frequent Itemsets.



Multistage Memory Mapping.

PCY: Key ideas

Each row contains, in potential, all the frequent/candidate itemsets.

Make efficient use of memory it’s important, use hash tables to select candidates.

Frequent Itemsets of size N, are a combination, without repetition of the individual items of N-1.

Why PCY?

?

Each row can be exploded in it subsets: processing can be parallelized at row level.

Why buckets ?

?

You can sum and obtain final frequency.

Why multiply for hashing, to generate just candidates?

?

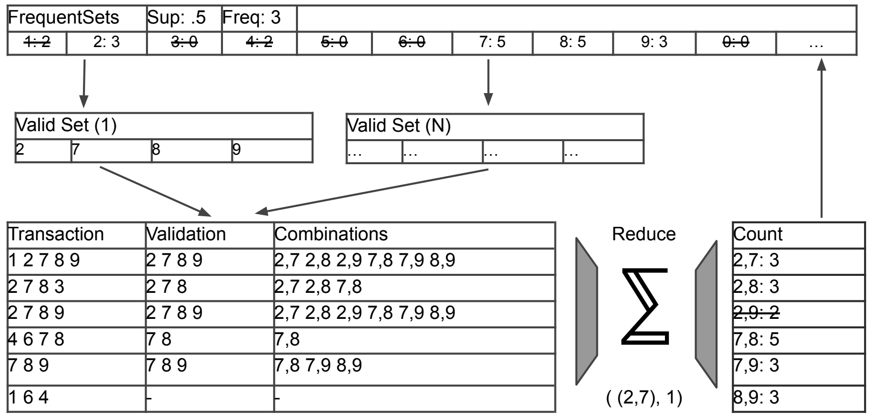
Remove every item you can before join and combinations.

Why joining huge tables?

PCY Bucketless

Steps

1. Count single transaction items
2. Obtain FrequentSets of size N=1.
3. Make the valid set, the FrequentSets on size N.
4. For each valid transaction line generate a list of actual sets of size N from the combination items of that row.
5. Each transaction line generates multiple sets, make them flat.
6. Reduce by key and filter.
7. Obtain Frequent Itemsets.



FrequentSets		Sup: .5	Freq: 3							
~~1: 2~~	2: 3	~~3: 0~~	~~4: 2~~	~~5: 0~~	~~6: 0~~	7: 5	8: 5	9: 3	~~0: 0~~	…

Valid Set (1)			
2	7	8	9

Valid Set (N)			
…	…	…	…

Transaction	Validation	Combinations
1 2 7 8 9	2 7 8 9	2,7 2,8 2,9 7,8 7,9 8,9
2 7 8 3	2 7 8	2,7 2,8 7,8
2 7 8 9	2 7 8 9	2,7 2,8 2,9 7,8 7,9 8,9
4 6 7 8	7 8	7,8
7 8 9	7 8 9	7,8 7,9 8,9
1 6 4	-	-

Count
2,7: 3
2,8: 3
~~2,9: 2~~
7,8: 5
7,9: 3
8,9: 3

PCY Bucketless: Focus

Reduce Transactions

```
pairsCount = transactions.flatMap(lambda transaction: findPairs(transaction, validationSet, candidateSize))
pairsCount = pairsCount.reduceByKey(lambda x, y: x + y)
pairsCount = pairsCount.filter(lambda item: item[1] > min_frequency)
```

Find Pairs

```
def findPairs(transaction:list, validationSet:tuple, candidateSize:int):
   if len(transaction) < candidateSize: return None
   validItems = set(filter(lambda item: item in validationSet, transaction))
   if len(validItems) < candidateSize: return tuple()

   pairs = combinations(transaction, candidateSize)
   return tuple((pair, 1) for pair in pairs)
```

How far can I push it?

★ Why not consolidating the item removal? It will avoid redundant future checks.

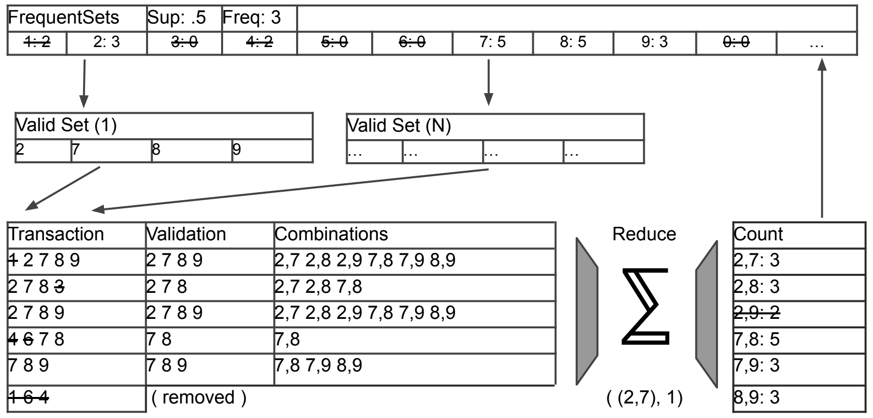
★ Keep calm and just count frequencies.

★ Progressive expansion of N allow to cheaply apply Erosion keeping the future expansions limited.

PCY Erode

Steps

1. Count single transaction items
2. Obtain FrequentSets of size N=1.
3. Make the ValidationSet, the FrequentSets on size N.
4. Remove each line with less then N items.
5. Remove from each transaction line the items not present in the itemsets.
6. Remove each line with less then N item.
7. For each transaction line generate a list of actual sets of size N from the combination items of that row.
8. Each transaction line generates multiple sets, make them flat.
9. Reduce by key and filter.
10. Filter candidates and obtain Frequent Itemsets.



FrequentSets		Sup: .5	Freq: 3							
~~1: 2~~	2: 3	~~3: 0~~	~~4: 2~~	~~5: 0~~	~~6: 0~~	7: 5	8: 5	9: 3	~~0: 0~~	…

Valid Set (1)			
2	7	8	9

Valid Set (N)			
…	…	…	…

Transaction	Validation	Combinations
~~1~~ 2 7 8 9	2 7 8 9	2,7 2,8 2,9 7,8 7,9 8,9
2 7 8 ~~3~~	2 7 8	2,7 2,8 7,8
2 7 8 9	2 7 8 9	2,7 2,8 2,9 7,8 7,9 8,9
~~4 6~~ 7 8	7 8	7,8
7 8 9	7 8 9	7,8 7,9 8,9
~~1 6 4~~	(removed)	

Count
2,7: 3
2,8: 3
~~2,9: 2~~
7,8: 5
7,9: 3
8,9: 3

PCY Erode: Focus

Reduce Transactions

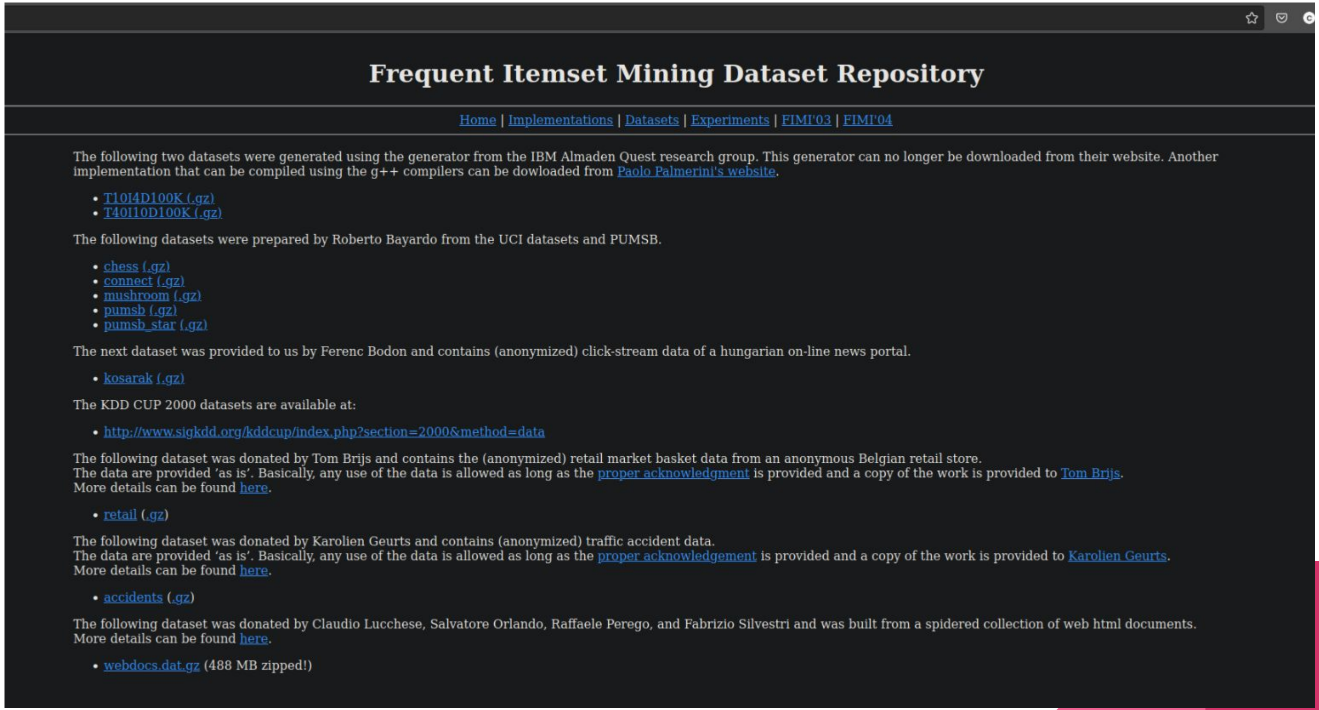
```python
transactions = reduceTransactions(transactions, validationSet, candidateSize)

def reduceTransactions(transactions: pyspark.RDD, validationSet, candidateSize:int)->pyspark.RDD:
   reducedSet = transactions.map(lambda transaction: filterTransactions(transaction, validationSet, candidateSize))
   reducedSet = reducedSet.filter(lambda transaction: transaction is not None)
   return reducedSet

def filterTransactions(transaction, validationSet, candidateSize:int):
   if len(transaction) < candidateSize: return None
   validItems = tuple(filter(lambda item: item in validationSet, transaction))
   if len(validItems) < candidateSize: return None
   return validItems
```

Find Pairs

```python
def findPairs(transaction:list|tuple, candidateSize:int) -> tuple:
   pairs = combinations(transaction, candidateSize)
   return tuple((pair, 1) for pair in pairs)
```

DATASETS

Dataset: Overview



Frequent Itemset Mining Dataset Repository

Home | Implementations | Datasets | Experiments | FIMI'03 | FIMI'04

The following two datasets were generated using the generator from the IBM Almaden Quest research group. This generator can no longer be downloaded from their website. Another implementation that can be compiled using the g++ compilers can be dowloaded from Paolo Palmerini's website.

- T10I4D100K (.gz)
- T40I10D100K (.gz)

The following datasets were prepared by Roberto Bayardo from the UCI datasets and PUMSB.

- chess (.gz)
- connect (.gz)
- mushroom (.gz)
- pumsb (.gz)
- pumsb_star (.gz)

The next dataset was provided to us by Ferenc Bodon and contains (anonymized) click-stream data of a hungarian on-line news portal.

- kosarak (.gz)

The KDD CUP 2000 datasets are available at:

- http://www.sigkdd.org/kddcup/index.php?section=2000&method=data

The following dataset was donated by Tom Brijs and contains the (anonymized) retail market basket data from an anonymous Belgian retail store.
The data are provided 'as is'. Basically, any use of the data is allowed as long as the proper acknowledgment is provided and a copy of the work is provided to Tom Brijs.
More details can be found here.

- retail (.gz)

The following dataset was donated by Karolien Geurts and contains (anonymized) traffic accident data.
The data are provided 'as is'. Basically, any use of the data is allowed as long as the proper acknowledgement is provided and a copy of the work is provided to Karolien Geurts.
More details can be found here.

- accidents (.gz)

The following dataset was donated by Claudio Lucchese, Salvatore Orlando, Raffaele Perego, and Fabrizio Silvestri and was built from a spidered collection of web html documents.
More details can be found here.

- webdocs.dat.gz (488 MB zipped!)

Dataset: Overview

				Support		
Name	**Size**	**Lines**	**Num Items**	**AVG**	**MAX**	**AVG Items**
Mushroom	570.408	8.124	119	0.193	1.000	23
Retail	4.167.490	88.162	16.470	0.001	0.575	10
50K10	2.008.453	50.000	869	0.012	0.078	10
25K40	3.869.620	25.000	942	0.042	0.288	40
100K10	4.022.055	100.000	870	0.0116	0.078	10
100K40	15.478.113	100.000	942	0.042	0.287	40

Dataset: Mushroom

General

Size	**570.408**
Num lines	**8.124**
Num Items	**119**
Support avg	**0.193**
Support max	**1**

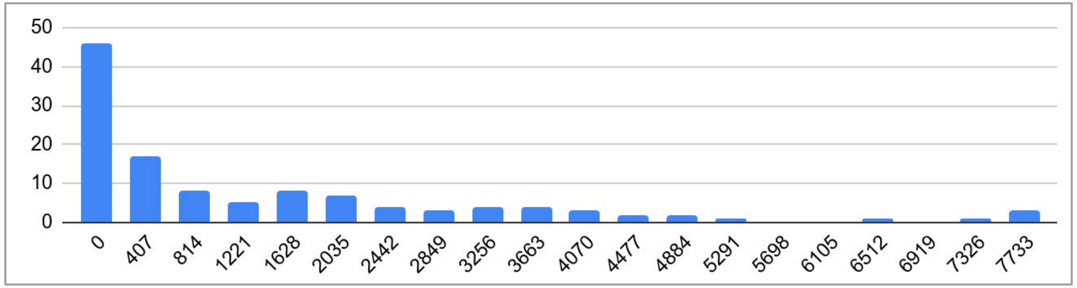
Frequency

Freq min	**4**
Freq max	**8.124**
Freq avg	**1.570**

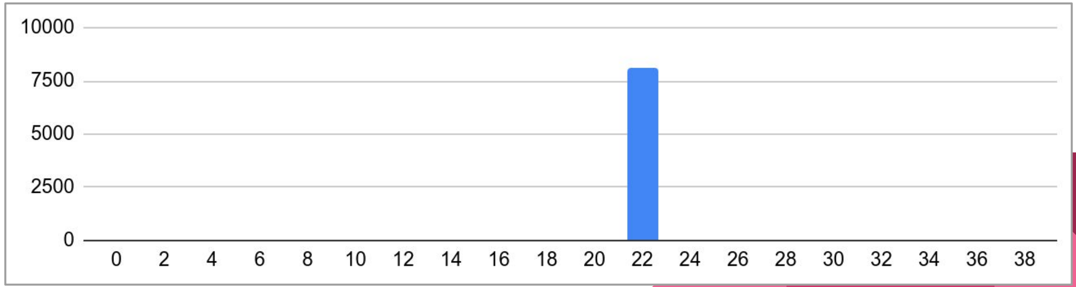
Transactions

Items min	**23**
items max	**23**
items avg	**23**

Items frequencies



Items per transaction



Dataset: Retail

General

Size	**4.167.490**
Num lines	**88.162**
Num Items	**16.470**
Support avg	**0.001**
Support max	**0.575**

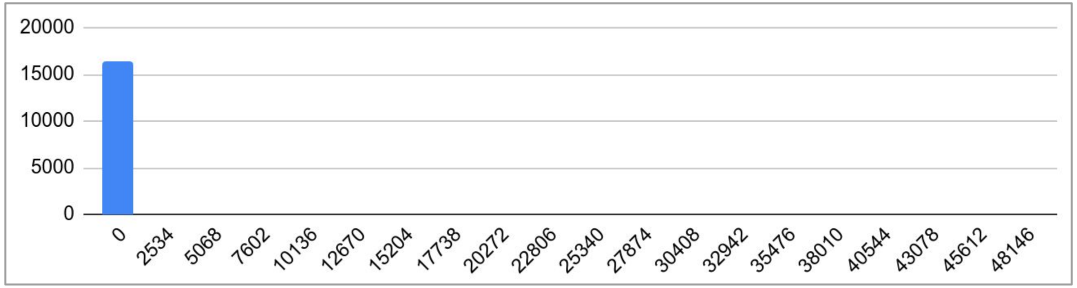
Frequency

Freq min	**1**
Freq max	**50.675**
Freq avg	**55**

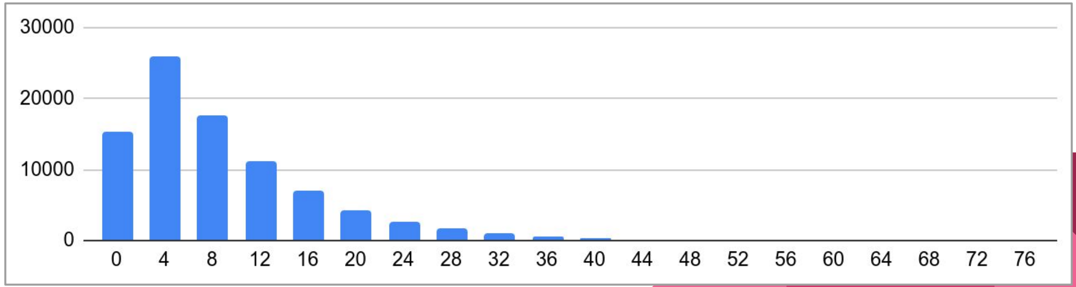
Transactions

Items min	**1**
items max	**76**
items avg	**10**

Items frequencies



Items per transaction



Dataset: 50K10

General

Size	**2.008.453**
Num lines	**50.000**
Num Items	**869**
Support avg	**0.012**
Support max	**0.078**

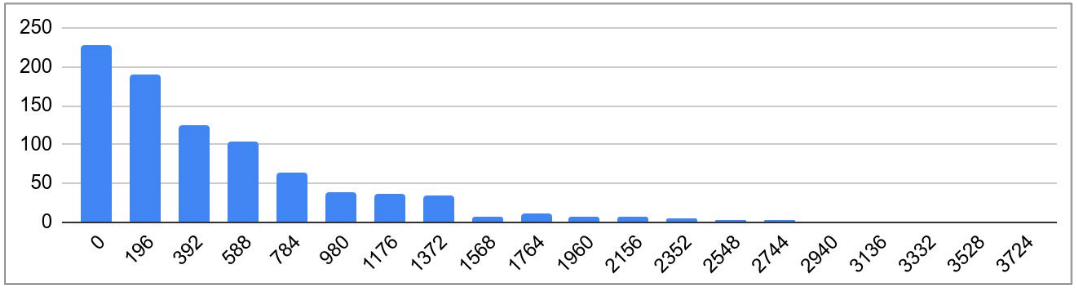
Frequency

Freq min	**2**
Freq max	**3.905**
Freq avg	**581**

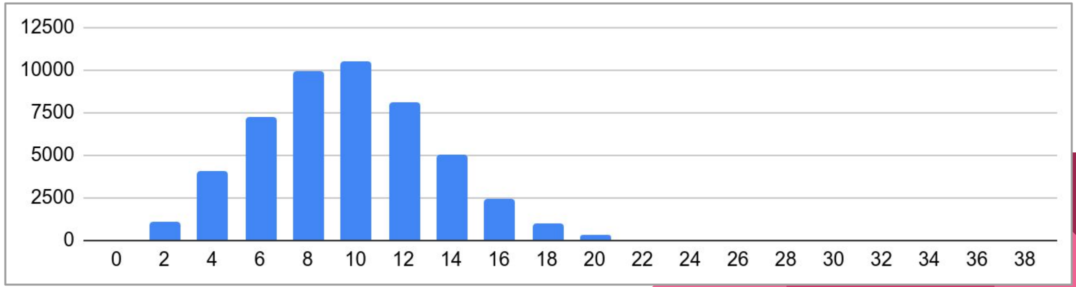
Transactions

Items min	**1**
items max	**29**
items avg	**10**

Items frequencies



Items per transaction



Dataset: 25K40

General

Size	**3.869.620**
Num lines	**25.000**
Num Items	**942**
Support avg	**0.042**
Support max	**0.288**

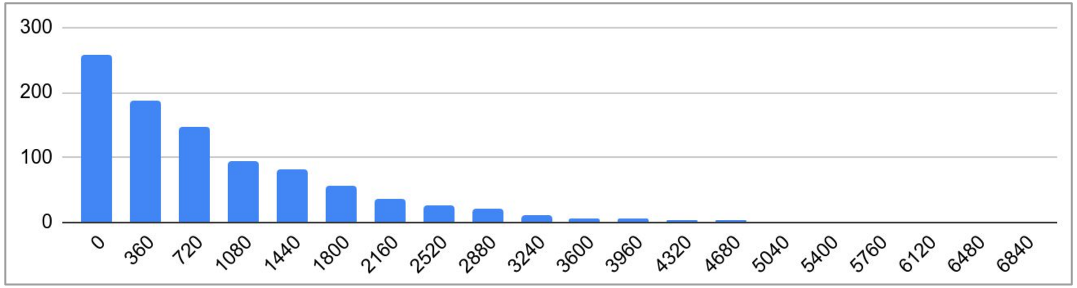
Frequency

Freq min	**1**
Freq max	**7188**
Freq avg	**1051**

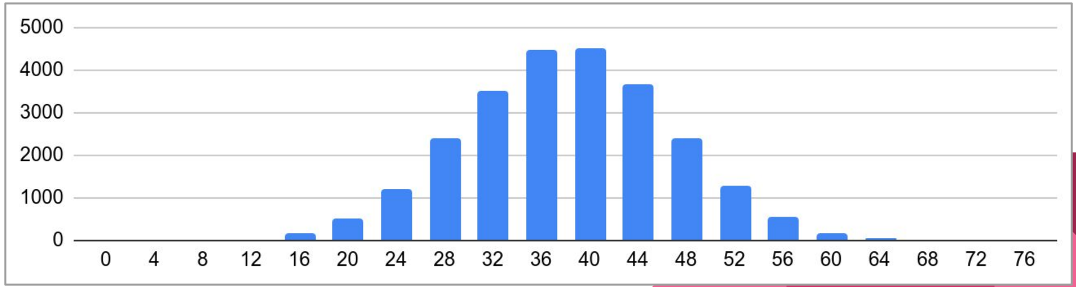
Transactions

Items min	**9**
items max	**72**
items avg	**40**

Items frequencies



Items per transaction



Dataset: 100K10

General

Size	**4.022.055**
Num lines	**100.000**
Num Items	**870**
Support avg	**0.012**
Support max	**0.078**

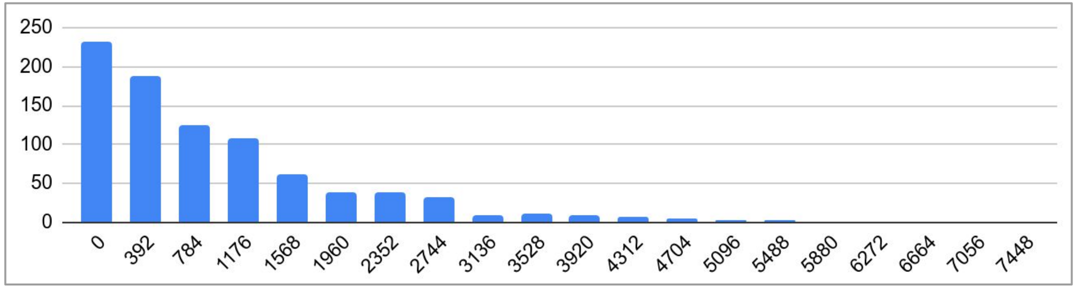
Frequency

Freq min	**1**
Freq max	**7828**
Freq avg	**1161**

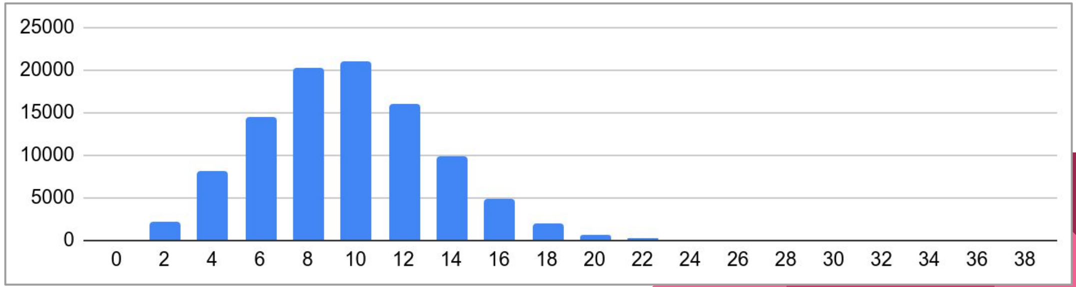
Transactions

Items min	**1**
items max	**29**
items avg	**10**

Items frequencies



Items per transaction



Dataset: 100K40

General

Size	**15.478.113**
Num lines	**100.000**
Num Items	**942**
Support avg	**0.042**
Support max	**0.287**

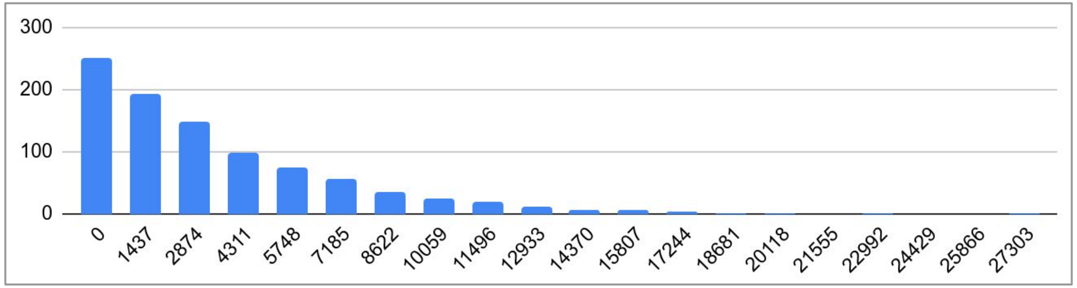
Frequency

Freq min	**5**
Freq max	**28738**
Freq avg	**4204**

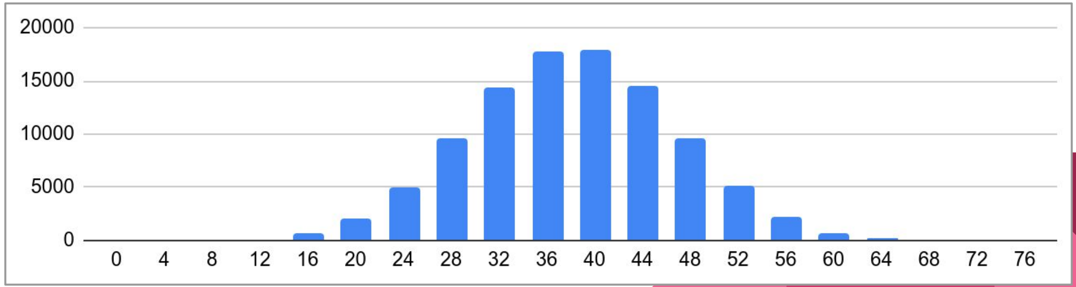
Transactions

Items min	**4**
items max	**77**
items avg	**40**

Items frequencies



Items per transaction



RESULTS

Results:

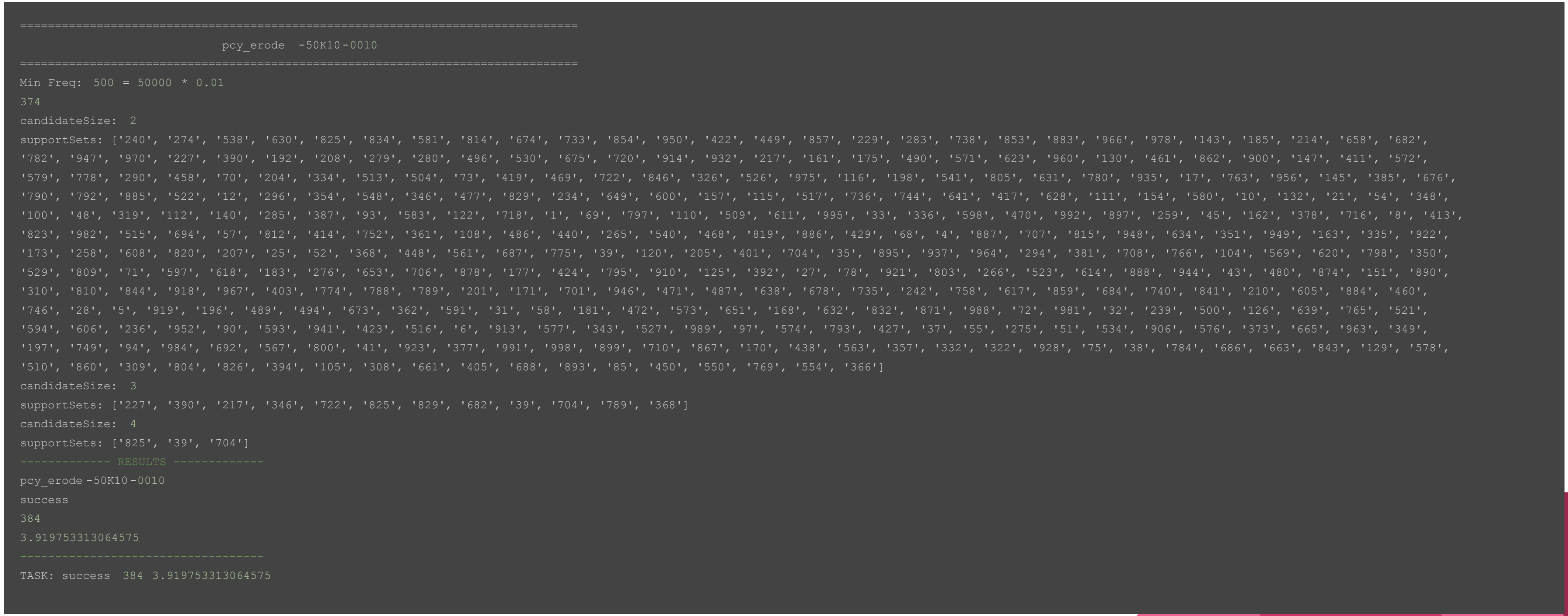
			apriori				
	Support	Itemset	py	spark	pcy	pcy erode	son
mushroom	0.9	**9**	**6**	**13**	**6**	**3**	**8**
	0.7	**31**	**18**	**111**	**164**	**4**	**9**
	0.5	**153**	**81**	**136**	**1.417**	**5**	**16**
	0.3	**2735**	**1.423**	OOM	**24.399**	**93**	**140**
retail	0.5	**1**	**531**	**96**	**3**	**3**	**60**
	0.3	**3**	**1.577**	**253**	**7**	**3**	**98**
	0.1	**9**	**4.803**	**723**	**140**	**3**	**221**
25K40	0.1	**81**	**1.740**	**254**	**9**	**3**	**137**
100K40	0.1	**82**	**7.115**	**1.040**	**26**	**6**	**940**
50K10	0.01	**384**	- - -	- - -	- - -	**4**	**426**
	0.03	**58**	- - -	- - -	- - -	**2**	**94**
	0.06	**4**	- - -	- - -	- - -	**2**	**31**
retail	0.005	**580**	- - -	- - -	- - -	**7**	**12.298**
	0.05	**16**	- - -	- - -	- - -	**3**	**339**

Results: Cost of a zero

		apriori				
		py	spark	pcy	pcy erode	son
retail	0.7	**1**	**2**	**3**	**2**	**41**
	0.9	**1**	**2**	**2**	**2**	**41**
100K10	0.1	**1**	**2**	**2**	**2**	**46**
	0.3	**1**	**2**	**2**	**2**	**45**
	0.5	**1**	**2**	**3**	**2**	**44**
	0.7	**1**	**2**	**2**	**2**	**45**
	0.9	**1**	**2**	**2**	**2**	**44**
100K40	0.3	**2**	**3**	**3**	**3**	**45**
	0.5	**2**	**3**	**3**	**3**	**45**
	0.7	**2**	**3**	**3**	**3**	**45**
	0.9	**2**	**3**	**3**	**3**	**45**

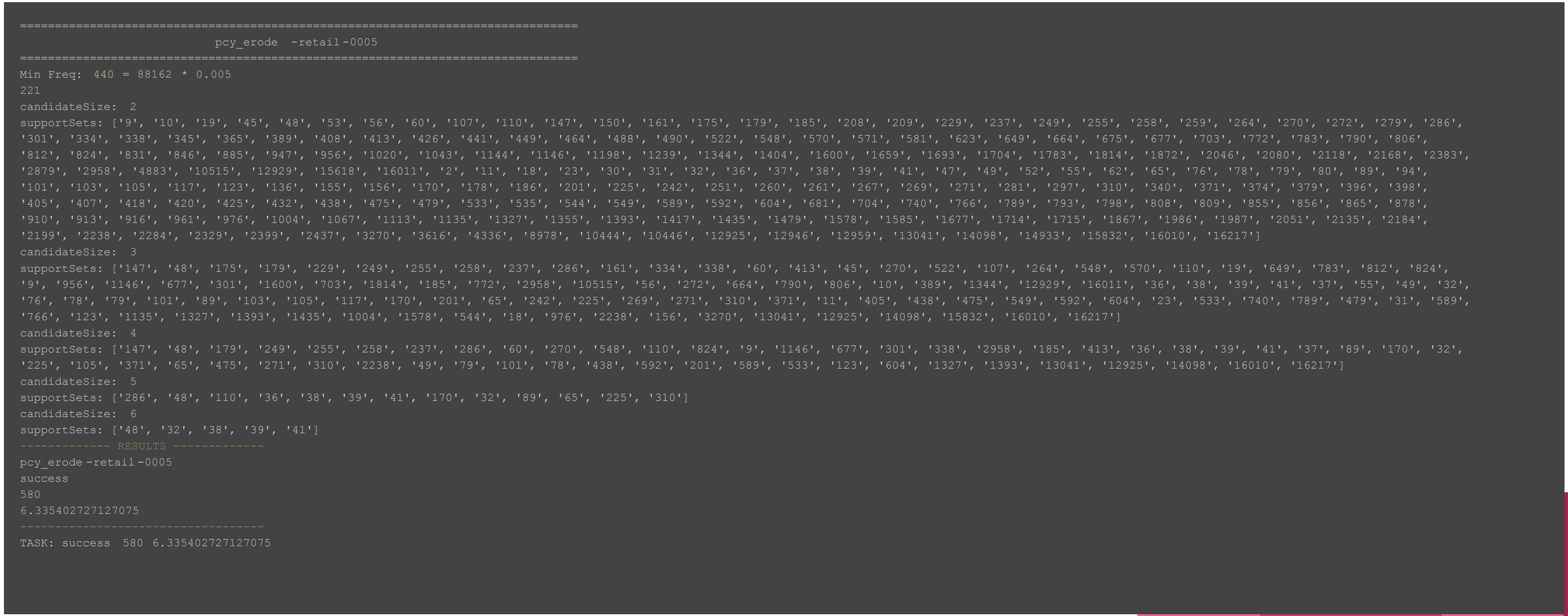
		apriori				
		py	spark	pcy	pcy erode	son
50K10	0.1	**1**	**2**	**2**	**2**	**24**
	0.3	**1**	**2**	**2**	**2**	**24**
	0.3	**1**	**2**	**2**	**2**	**24**
	0.7	**1**	**2**	**2**	**2**	**25**
	0.9	**1**	**2**	**2**	**2**	**25**
100K40	0.3	**2**	**3**	**3**	**3**	**45**
	0.5	**2**	**3**	**3**	**3**	**45**
	0.7	**2**	**3**	**3**	**3**	**45**
	0.9	**2**	**3**	**3**	**3**	**45**

Erosion at work: 50K10 @ 0.01 (500)



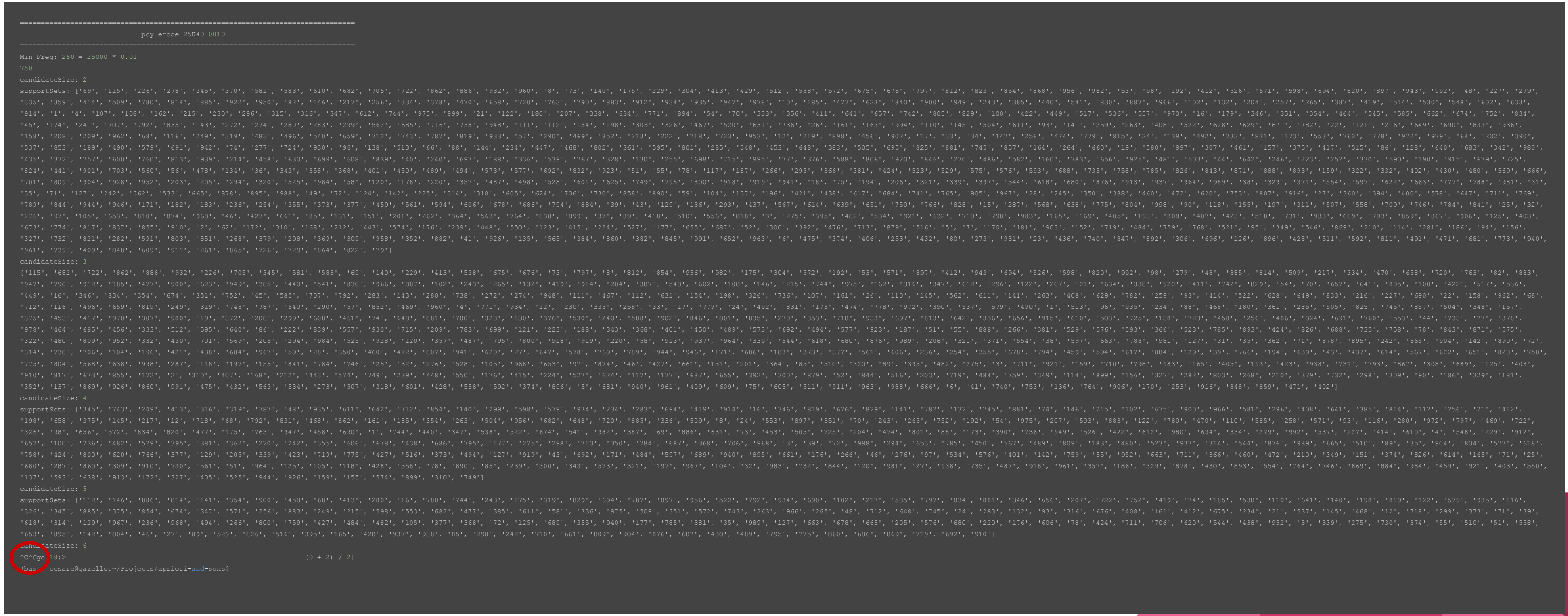
```
==============================================================================
                            pcy_erode  -50K10-0010
==============================================================================
Min Freq:  500 = 50000 * 0.01
374
candidateSize:  2
supportSets: ['240', '274', '538', '630', '825', '834', '581', '814', '674', '733', '854', '950', '422', '449', '857', '229', '283', '738', '853', '883', '966', '978', '143', '185', '214', '658', '682',
'782', '947', '970', '227', '390', '192', '208', '279', '280', '496', '530', '675', '720', '914', '932', '217', '161', '175', '490', '571', '623', '960', '130', '461', '862', '900', '147', '411', '572',
'579', '778', '290', '458', '70', '204', '334', '513', '504', '73', '419', '469', '722', '846', '326', '526', '975', '116', '198', '541', '805', '631', '780', '935', '17', '763', '956', '145', '385', '676',
'790', '792', '885', '522', '12', '296', '354', '548', '346', '477', '829', '234', '649', '600', '157', '115', '517', '736', '744', '641', '417', '628', '111', '154', '580', '10', '132', '21', '54', '348',
'100', '48', '319', '112', '140', '285', '387', '93', '583', '122', '718', '1', '69', '797', '110', '509', '611', '995', '33', '336', '598', '470', '992', '897', '259', '45', '162', '378', '716', '8', '413',
'823', '982', '515', '694', '57', '812', '414', '752', '361', '108', '486', '440', '265', '540', '468', '819', '886', '429', '68', '4', '887', '707', '815', '948', '634', '351', '949', '163', '335', '922',
'173', '258', '608', '820', '207', '25', '52', '368', '448', '561', '687', '775', '39', '120', '205', '401', '704', '35', '895', '937', '964', '294', '381', '708', '766', '104', '569', '620', '798', '350',
'529', '809', '71', '597', '618', '183', '276', '653', '706', '878', '177', '424', '795', '910', '125', '392', '27', '78', '921', '803', '266', '523', '614', '888', '944', '43', '480', '874', '151', '890',
'310', '810', '844', '918', '967', '403', '774', '788', '789', '201', '171', '701', '946', '471', '487', '638', '678', '735', '242', '758', '617', '859', '684', '740', '841', '210', '605', '884', '460',
'746', '28', '5', '919', '196', '489', '494', '673', '362', '591', '31', '58', '181', '472', '573', '651', '168', '632', '832', '871', '988', '72', '981', '32', '239', '500', '126', '639', '765', '521',
'594', '606', '236', '952', '90', '593', '941', '423', '516', '6', '913', '577', '343', '527', '989', '97', '574', '793', '427', '37', '55', '275', '51', '534', '906', '576', '373', '665', '963', '349',
'197', '749', '94', '984', '692', '567', '800', '41', '923', '377', '991', '998', '899', '710', '867', '170', '438', '563', '357', '332', '322', '928', '75', '38', '784', '686', '663', '843', '129', '578',
'510', '860', '309', '804', '826', '394', '105', '308', '661', '405', '688', '893', '85', '450', '550', '769', '554', '366']
candidateSize:  3
supportSets: ['227', '390', '217', '346', '722', '825', '829', '682', '39', '704', '789', '368']
candidateSize:  4
supportSets: ['825', '39', '704']
------------- RESULTS -------------
pcy_erode-50K10-0010
success
384
3.919753313064575
-----------------------------------
TASK: success  384  3.919753313064575
```

Erosion at work: retail @ 0.005 (440)



```
================================================================================
                            pcy_erode  -retail-0005
================================================================================
Min Freq:  440 = 88162 * 0.005
221
candidateSize:  2
supportSets: ['9', '10', '19', '45', '48', '53', '56', '60', '107', '110', '147', '150', '161', '175', '179', '185', '208', '209', '229', '237', '249', '255', '258', '259', '264', '270', '272', '279', '286',
'301', '334', '338', '345', '365', '389', '408', '413', '426', '441', '449', '464', '488', '490', '522', '548', '570', '571', '581', '623', '649', '664', '675', '677', '703', '772', '783', '790', '806',
'812', '824', '831', '846', '885', '947', '956', '1020', '1043', '1144', '1146', '1198', '1239', '1344', '1404', '1600', '1659', '1693', '1704', '1783', '1814', '1872', '2046', '2080', '2118', '2168', '2383',
'2879', '2958', '4883', '10515', '12929', '15618', '16011', '2', '11', '18', '23', '30', '31', '32', '36', '37', '38', '39', '41', '47', '49', '52', '55', '62', '65', '76', '78', '79', '80', '89', '94',
'101', '103', '105', '117', '123', '136', '155', '156', '170', '178', '186', '201', '225', '242', '251', '260', '261', '267', '269', '271', '281', '297', '310', '340', '371', '374', '379', '396', '398',
'405', '407', '418', '420', '425', '432', '438', '475', '479', '533', '535', '544', '549', '589', '592', '604', '681', '704', '740', '766', '789', '793', '798', '808', '809', '855', '856', '865', '878',
'910', '913', '916', '961', '976', '1004', '1067', '1113', '1135', '1327', '1355', '1393', '1417', '1435', '1479', '1578', '1585', '1677', '1714', '1715', '1867', '1986', '1987', '2051', '2135', '2184',
'2199', '2238', '2284', '2329', '2399', '2437', '3270', '3616', '4336', '8978', '10444', '10446', '12925', '12946', '12959', '13041', '14098', '14933', '15832', '16010', '16217']
candidateSize:  3
supportSets: ['147', '48', '175', '179', '229', '249', '255', '258', '237', '286', '161', '334', '338', '60', '413', '45', '270', '522', '107', '264', '548', '570', '110', '19', '649', '783', '812', '824',
'9', '956', '1146', '677', '301', '1600', '703', '1814', '185', '772', '2958', '10515', '56', '272', '664', '790', '806', '10', '389', '1344', '12929', '16011', '36', '38', '39', '41', '37', '55', '49', '32',
'76', '78', '79', '101', '89', '103', '105', '117', '170', '201', '65', '242', '225', '269', '271', '310', '371', '11', '405', '438', '475', '549', '592', '604', '23', '533', '740', '789', '479', '31', '589',
'766', '123', '1135', '1327', '1393', '1435', '1004', '1578', '544', '18', '976', '2238', '156', '3270', '13041', '12925', '14098', '15832', '16010', '16217']
candidateSize:  4
supportSets: ['147', '48', '179', '249', '255', '258', '237', '286', '60', '270', '548', '110', '824', '9', '1146', '677', '301', '338', '2958', '185', '413', '36', '38', '39', '41', '37', '89', '170', '32',
'225', '105', '371', '65', '475', '271', '310', '2238', '49', '79', '101', '78', '438', '592', '201', '589', '533', '123', '604', '1327', '1393', '13041', '12925', '14098', '16010', '16217']
candidateSize:  5
supportSets: ['286', '48', '110', '36', '38', '39', '41', '170', '32', '89', '65', '225', '310']
candidateSize:  6
supportSets: ['48', '32', '38', '39', '41']
------------- RESULTS -------------
pcy_erode -retail -0005
success
580
6.335402727127075
-----------------------------------
TASK: success  580  6.335402727127075
```

Erosion at work: 25K40 @ 0.01 (250)



```
================================================================================
                             pcy_erode-25K40-0010
================================================================================
Min Freq: 250 = 25000 * 0.01
750
candidateSize: 2
supportSets: ['69', '115', '226', '278', '345', '370', '581', '583', '610', '682', '705', '722', '862', '886', '932', '960', '8', '73', '140', '175', '229', '304', '413', '429', '512', '538', '572', '675', '676', '797', '812', '823', '854', '868', '956', '982', '53', '98', '192', '412', '526', '571', '598', '694', '820', '897', '943', '992', '48', '227', '279',
'335', '359', '414', '509', '780', '814', '885', '922', '950', '82', '146', '217', '256', '334', '378', '470', '658', '720', '763', '790', '883', '912', '934', '935', '947', '978', '10', '185', '477', '623', '840', '900', '949', '243', '385', '440', '541', '830', '887', '966', '102', '132', '204', '257', '265', '387', '419', '514', '530', '548', '602', '633',
'914', '1', '4', '107', '108', '162', '215', '230', '296', '315', '316', '347', '612', '744', '975', '999', '21', '122', '180', '207', '338', '634', '771', '894', '54', '70', '333', '356', '411', '641', '657', '742', '805', '829', '100', '422', '449', '517', '536', '557', '970', '16', '179', '346', '351', '354', '464', '545', '585', '662', '674', '752', '834',
'45', '174', '241', '707', '792', '835', '143', '272', '274', '280', '283', '299', '562', '685', '716', '738', '948', '111', '112', '154', '198', '303', '326', '467', '520', '631', '736', '26', '161', '163', '994', '110', '145', '504', '611', '93', '141', '259', '263', '408', '522', '628', '629', '671', '782', '22', '121', '216', '649', '690', '833', '936',
'158', '208', '209', '962', '68', '116', '249', '319', '483', '496', '540', '659', '712', '743', '787', '819', '933', '57', '290', '469', '852', '213', '222', '718', '723', '953', '12', '219', '898', '456', '902', '17', '33', '34', '147', '258', '474', '779', '815', '24', '139', '492', '733', '831', '173', '553', '762', '778', '972', '979', '64', '202', '390',
'537', '853', '189', '490', '579', '691', '942', '74', '277', '724', '930', '96', '138', '513', '66', '88', '144', '234', '447', '468', '802', '361', '595', '801', '285', '348', '453', '648', '383', '505', '695', '825', '881', '745', '857', '164', '264', '660', '19', '580', '997', '307', '461', '157', '375', '417', '515', '86', '128', '640', '683', '342', '980',
'435', '372', '757', '600', '760', '813', '939', '214', '458', '630', '699', '608', '839', '40', '240', '697', '188', '336', '539', '767', '328', '130', '255', '698', '715', '995', '77', '376', '588', '806', '920', '846', '270', '486', '582', '160', '783', '656', '925', '481', '503', '44', '642', '246', '223', '252', '330', '590', '190', '915', '679', '725',
'824', '441', '901', '703', '560', '56', '478', '134', '36', '343', '358', '368', '401', '450', '489', '494', '573', '577', '692', '832', '923', '51', '55', '78', '117', '187', '266', '295', '366', '381', '424', '523', '529', '575', '576', '593', '688', '735', '758', '785', '826', '843', '871', '888', '893', '159', '322', '332', '402', '430', '480', '569', '666',
'701', '809', '904', '928', '952', '203', '205', '294', '320', '525', '984', '58', '120', '178', '220', '357', '487', '498', '528', '601', '625', '749', '795', '800', '918', '919', '941', '18', '75', '194', '206', '321', '339', '397', '544', '618', '680', '876', '913', '937', '964', '989', '38', '329', '371', '554', '597', '622', '663', '777', '788', '981', '31',
'35', '71', '127', '242', '362', '533', '665', '878', '895', '988', '49', '72', '124', '142', '225', '314', '318', '605', '624', '706', '730', '858', '890', '59', '104', '137', '196', '421', '438', '617', '684', '741', '765', '905', '967', '28', '245', '350', '388', '460', '472', '620', '753', '807', '916', '27', '360', '394', '400', '578', '647', '711', '769',
'789', '844', '944', '946', '171', '182', '183', '236', '254', '355', '373', '377', '459', '561', '594', '606', '678', '686', '794', '884', '39', '43', '129', '136', '293', '437', '567', '614', '639', '651', '750', '766', '828', '15', '287', '568', '638', '775', '804', '998', '90', '118', '155', '197', '311', '507', '558', '709', '746', '784', '841', '25', '32',
'276', '97', '105', '653', '810', '874', '968', '46', '427', '661', '85', '131', '151', '201', '262', '364', '563', '764', '838', '899', '37', '89', '418', '510', '556', '818', '3', '275', '395', '482', '534', '921', '632', '710', '798', '983', '165', '169', '405', '193', '308', '407', '423', '518', '731', '938', '689', '793', '859', '867', '906', '125', '403',
'673', '774', '817', '837', '855', '910', '2', '62', '172', '310', '168', '212', '443', '574', '176', '239', '448', '550', '123', '415', '224', '527', '177', '655', '687', '52', '300', '392', '476', '713', '879', '516', '5', '7', '170', '181', '903', '152', '719', '484', '759', '768', '521', '95', '349', '546', '869', '210', '114', '281', '186', '94', '156',
'327', '732', '821', '282', '591', '803', '851', '268', '379', '298', '369', '309', '958', '352', '882', '41', '926', '135', '565', '384', '860', '382', '845', '991', '652', '963', '6', '475', '374', '406', '253', '432', '80', '273', '931', '23', '436', '740', '847', '892', '306', '696', '126', '896', '428', '511', '592', '811', '491', '471', '681', '773', '940',
'961', '739', '409', '848', '609', '911', '261', '865', '726', '729', '864', '822', '79']
candidateSize: 3
['115', '682', '722', '862', '886', '932', '226', '705', '345', '581', '583', '69', '140', '229', '413', '538', '675', '676', '73', '797', '8', '812', '854', '956', '982', '175', '304', '572', '192', '53', '571', '897', '412', '943', '694', '526', '598', '820', '992', '98', '279', '48', '885', '814', '509', '217', '334', '470', '658', '720', '763', '82', '883',
'947', '790', '912', '185', '477', '900', '623', '949', '385', '440', '541', '830', '966', '887', '102', '243', '265', '132', '419', '914', '204', '387', '548', '602', '108', '146', '215', '744', '975', '162', '316', '347', '612', '296', '122', '207', '21', '634', '338', '922', '411', '742', '829', '54', '70', '657', '641', '805', '100', '422', '517', '536',
'449', '16', '346', '834', '354', '674', '351', '752', '45', '585', '707', '792', '283', '143', '280', '738', '272', '274', '948', '111', '467', '112', '631', '154', '198', '326', '736', '107', '161', '26', '110', '145', '562', '611', '141', '263', '408', '629', '782', '259', '93', '414', '522', '628', '649', '833', '216', '227', '690', '22', '158', '962', '68',
'712', '116', '496', '659', '819', '249', '319', '743', '787', '540', '290', '57', '852', '469', '960', '4', '771', '934', '12', '230', '335', '258', '33', '17', '779', '24', '492', '831', '173', '474', '778', '972', '390', '537', '579', '490', '1', '513', '96', '935', '234', '88', '468', '180', '361', '285', '505', '825', '745', '857', '504', '348', '157',
'375', '453', '417', '970', '307', '980', '19', '372', '208', '299', '608', '461', '74', '648', '881', '780', '328', '130', '376', '530', '240', '588', '902', '846', '801', '835', '270', '853', '718', '933', '697', '813', '642', '336', '656', '915', '610', '503', '725', '138', '723', '458', '256', '486', '824', '691', '760', '553', '44', '733', '77', '378',
'978', '464', '685', '456', '333', '512', '595', '640', '86', '222', '839', '557', '930', '715', '209', '783', '699', '121', '223', '188', '343', '368', '401', '450', '489', '573', '692', '494', '577', '923', '187', '51', '55', '888', '266', '381', '529', '576', '593', '366', '523', '785', '893', '424', '826', '688', '735', '758', '78', '843', '871', '575',
'322', '480', '809', '952', '332', '430', '701', '569', '205', '294', '984', '525', '928', '120', '357', '487', '795', '800', '918', '919', '220', '58', '913', '937', '964', '339', '544', '618', '680', '876', '989', '206', '321', '371', '554', '38', '597', '663', '788', '981', '127', '31', '35', '362', '71', '878', '895', '242', '665', '904', '142', '890', '72',
'314', '730', '706', '104', '196', '421', '438', '684', '967', '59', '28', '350', '460', '472', '807', '941', '620', '27', '647', '578', '769', '789', '944', '946', '171', '686', '183', '373', '377', '561', '606', '236', '254', '355', '678', '794', '459', '594', '617', '884', '129', '39', '766', '194', '639', '43', '437', '614', '567', '622', '651', '828', '750',
'775', '804', '568', '638', '998', '287', '118', '197', '155', '841', '784', '746', '25', '32', '276', '528', '105', '968', '653', '97', '874', '46', '427', '661', '151', '201', '364', '85', '510', '320', '89', '395', '482', '275', '3', '711', '921', '159', '710', '798', '983', '165', '405', '193', '423', '938', '731', '793', '867', '308', '689', '125', '403',
'910', '817', '673', '855', '172', '2', '310', '407', '168', '212', '443', '574', '749', '239', '448', '550', '176', '415', '224', '527', '624', '117', '177', '687', '655', '392', '300', '879', '52', '844', '516', '203', '719', '484', '759', '349', '114', '899', '156', '327', '282', '803', '268', '210', '379', '732', '298', '309', '90', '186', '329', '181',
'352', '137', '869', '926', '860', '991', '475', '432', '563', '534', '273', '507', '318', '601', '428', '558', '592', '374', '896', '5', '681', '940', '961', '409', '609', '75', '605', '511', '911', '963', '988', '666', '6', '41', '740', '753', '136', '764', '906', '170', '253', '916', '848', '859', '471', '402']
candidateSize: 4
supportSets: ['345', '743', '249', '413', '316', '319', '787', '48', '935', '611', '642', '712', '854', '140', '299', '598', '579', '934', '234', '283', '694', '419', '914', '16', '346', '819', '676', '829', '141', '782', '132', '745', '881', '74', '146', '215', '102', '675', '900', '966', '581', '296', '408', '641', '385', '814', '112', '256', '21', '412',
'198', '658', '375', '145', '217', '12', '718', '68', '792', '831', '468', '862', '161', '185', '354', '263', '504', '956', '682', '648', '720', '885', '336', '509', '8', '24', '553', '897', '351', '70', '243', '265', '752', '192', '54', '975', '207', '503', '883', '122', '780', '470', '110', '585', '258', '571', '93', '116', '280', '972', '797', '469', '722',
'326', '98', '656', '572', '834', '820', '477', '175', '763', '947', '458', '690', '1', '744', '440', '347', '538', '522', '674', '541', '982', '387', '69', '886', '631', '73', '453', '505', '725', '204', '474', '801', '88', '173', '390', '736', '949', '526', '422', '612', '980', '634', '334', '279', '992', '537', '227', '414', '610', '4', '548', '229', '912',
'657', '100', '236', '482', '529', '395', '381', '362', '220', '242', '355', '606', '678', '438', '686', '795', '177', '275', '298', '710', '350', '784', '687', '368', '706', '968', '3', '39', '72', '998', '294', '653', '785', '450', '567', '489', '809', '183', '480', '523', '937', '314', '544', '876', '989', '665', '510', '89', '35', '904', '804', '577', '618',
'758', '424', '800', '620', '766', '377', '129', '205', '339', '423', '719', '775', '427', '516', '373', '494', '127', '919', '43', '692', '171', '484', '597', '689', '940', '895', '661', '176', '266', '46', '276', '97', '534', '576', '401', '142', '759', '55', '952', '663', '711', '366', '460', '472', '210', '349', '151', '374', '826', '614', '165', '71', '25',
'680', '287', '860', '309', '910', '730', '561', '51', '964', '125', '105', '118', '428', '558', '78', '890', '85', '239', '300', '343', '573', '321', '197', '967', '104', '32', '983', '732', '844', '120', '981', '27', '938', '735', '487', '918', '961', '357', '186', '329', '878', '430', '893', '554', '764', '746', '869', '884', '984', '459', '921', '403', '550',
'137', '593', '638', '913', '172', '327', '405', '525', '944', '926', '159', '155', '574', '899', '310', '749']
candidateSize: 5
supportSets: ['112', '146', '886', '814', '141', '354', '900', '458', '68', '413', '280', '16', '780', '744', '243', '175', '319', '829', '694', '787', '897', '956', '522', '792', '934', '690', '102', '217', '585', '797', '834', '881', '346', '656', '207', '722', '752', '419', '74', '185', '538', '110', '641', '140', '198', '819', '122', '579', '935', '116',
'326', '345', '885', '375', '854', '674', '347', '571', '256', '883', '249', '215', '598', '553', '682', '477', '385', '611', '581', '336', '975', '509', '351', '572', '743', '263', '966', '265', '48', '712', '648', '745', '24', '283', '132', '93', '316', '676', '408', '161', '412', '675', '234', '21', '537', '145', '468', '12', '718', '299', '373', '71', '39',
'618', '314', '129', '967', '236', '968', '494', '266', '800', '759', '427', '484', '482', '105', '377', '368', '72', '125', '689', '355', '940', '177', '785', '381', '35', '989', '127', '663', '678', '665', '205', '576', '680', '220', '176', '606', '78', '424', '711', '706', '620', '544', '438', '952', '3', '339', '275', '730', '374', '55', '510', '51', '558',
'653', '895', '142', '804', '46', '27', '89', '529', '826', '516', '395', '165', '428', '937', '938', '85', '298', '242', '710', '661', '809', '904', '876', '687', '480', '489', '795', '775', '860', '686', '869', '719', '692', '910']
candidateSize: 6
^C^Cge 18:>                                            (0 + 2) / 2]
(base) cesare@gazelle:~/Projects/apriori-and-sons$
```

Other findings

Hadoop native libraries

```
23/07/02 14:44:47 WARN NativeCodeLoader:
Unable to load native-hadoop library for your platform... using builtin-java classes where applicable.
```

NO GAIN

Using Integers

```
def load(sc, data_path, sep=' ', useInt=True) -> pyspark.RDD:
   itemSets = sc.textFile(data_path)
   itemSets = itemSets.map(lambda line: line.strip().split(sep))
   if useInt:
       itemSets = itemSets.map(lambda line: tuple(map(lambda item: int(item), line)))
   return itemSets
```

-10%

		Sup	Strings	Integers	Time saved
mushroom	apriori spark	0.7	113	115	-0%
	pcy erode	0.3	93	77	-15%
	apriori python	0.5	81	69	-10%

(pure pyspark)

CONCLUSIONS

Conclusions

- <u>Always plot stats of your dataset as first step, will save tons of time down the line.</u>
- Curse of dimensionality:
    - Join and Catensian products should be avoid or limited.
    - Alternatives: in-transaction line combinations
    - *Better a Join then a loop: combinations can be an option but can become triky.*
- Filtering can help reducing the overall computation and if done cheaply doesn't decrease the performance in the worst case scenario.
- Performing an initial filtering even against the Frequent Itemset of size 1, which is anyway computed, can significantly reduce the number items per line, as well as overall lines, greatly decreasing.
- Finding and Verifying Candidate Itemsets can be both wasteful and very expensive.
- In-line combinations when used in combination with counting and filtering can be a scalable alternative to often wasteful cartesian products.
- PCY Erode shows that progressive erosion of the dataset and controlled expansions of the rows, an effective ways to keep under control faster-then-linear growths.

FUTURE

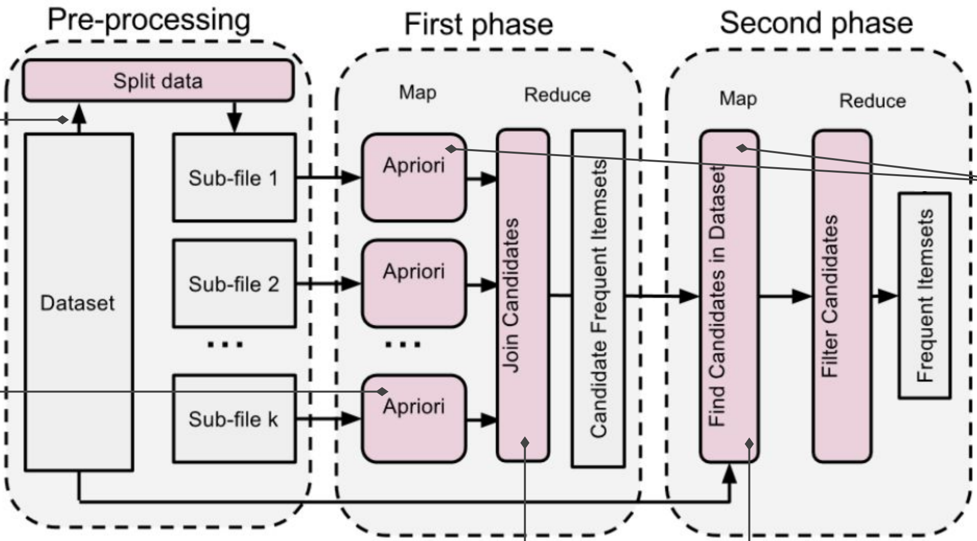
Future

- By the current results I foresee a potential for proving “PCY Erode” with even more precise line filtering and while removing the need of resorting to combinations, potentially reaching an almost guaranteed minimum expansions of each row. However the computational cost of implementing such an aggressive filtering has yet to be accessed and trade-offs has yet to be evaluated.

- Considering that “PCY Erode” doesn’t use the concept of buckets should be a good algorithm to replace apriori to discover Candidate Itemsets.

- By the current results I believe it should be possible to apply several optimizations also to the second phase of the SON algorithm by reusing the partitioning generated at the first step, combined with filtering and in-line expansion, to sensibly reduce the size of the cartesian product between the transaction and the candidate itemset.

Future: SON Erode



★ Find FrequentSet N = 1 and apply Erosion

★ Use PCY Bucketless or PCY Erode* instead.

★ Remove FrequentSet N = 1 from Candidates

★ Apply Erosion based on Candidates Sets

“Pig-Partitioning”

? Can I salvage some of the information from the buckets to optimize testing of candidates ?

* need to make a copy of the partition