I use an enumeration tree structure to store the projected database, at each node, name P, I maintain candidate extension items for node P, and a transaction database that only contains transactions that contain the current itemset of P and in every transaction, only those who are lexicographically greater than the current item of P are preserved. The current itemset of every node is generated by add one element in the candidate extension of its parent node to the current itemset of its parent node. The count of every node is achieved by counting transactions of its parent node that contain the element extended.

In the code, I used a two-dimensional ArrayList, which is a space expensive and stupid way, to store every projected database.

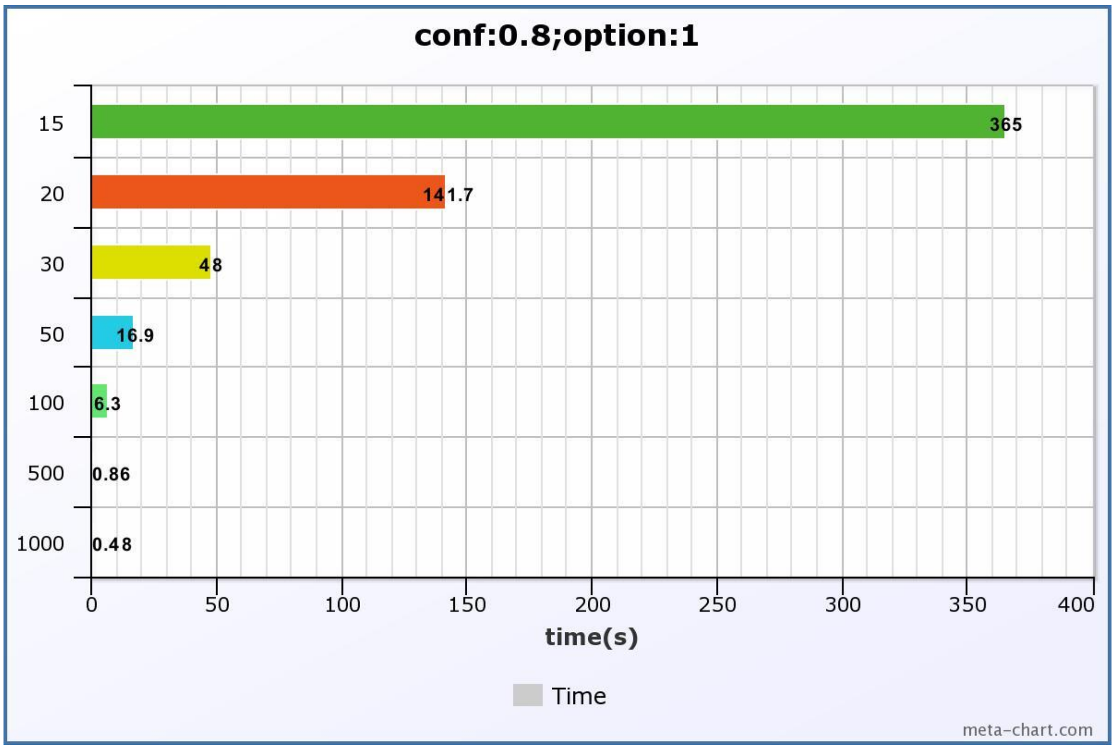
I just realized that sparse matrix might be a better choice to store the database. Each line of the input file is stored as an element in a m*2 int array(m is the number of lines in the input file, for each line, index 0 stores the transaction number, and index 1 stores the item). For each node maintain an indicator array that contains pointers to the first element of each transaction that contains the current itemset of node P, and we start from the first element to search for the candidate extension now we are looking at. If we find the extension item, the count of the node P increments. This might be a quicker way to do the projection, but since I start the project assignment late, time is not enough for me to implement it.

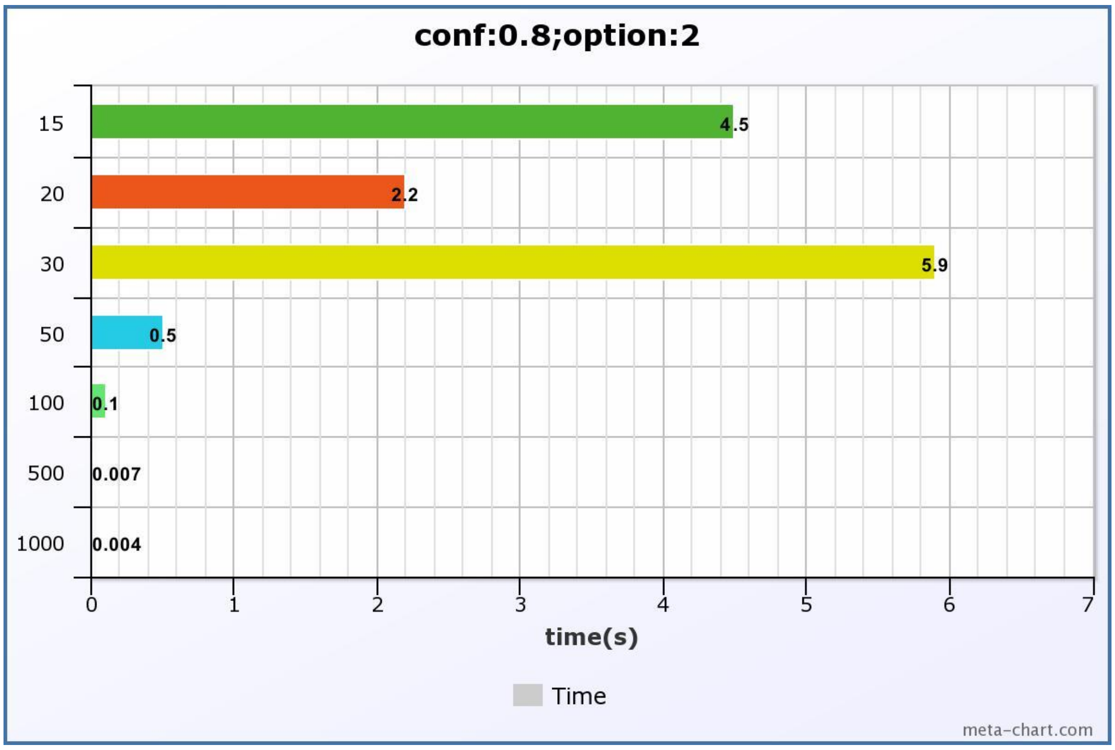
In order to improve efficiency, I tried the buckets method mentioned in Charu's book in section 4.4.3.2. But it doesn't work better than the previous one. Maybe It's my bad

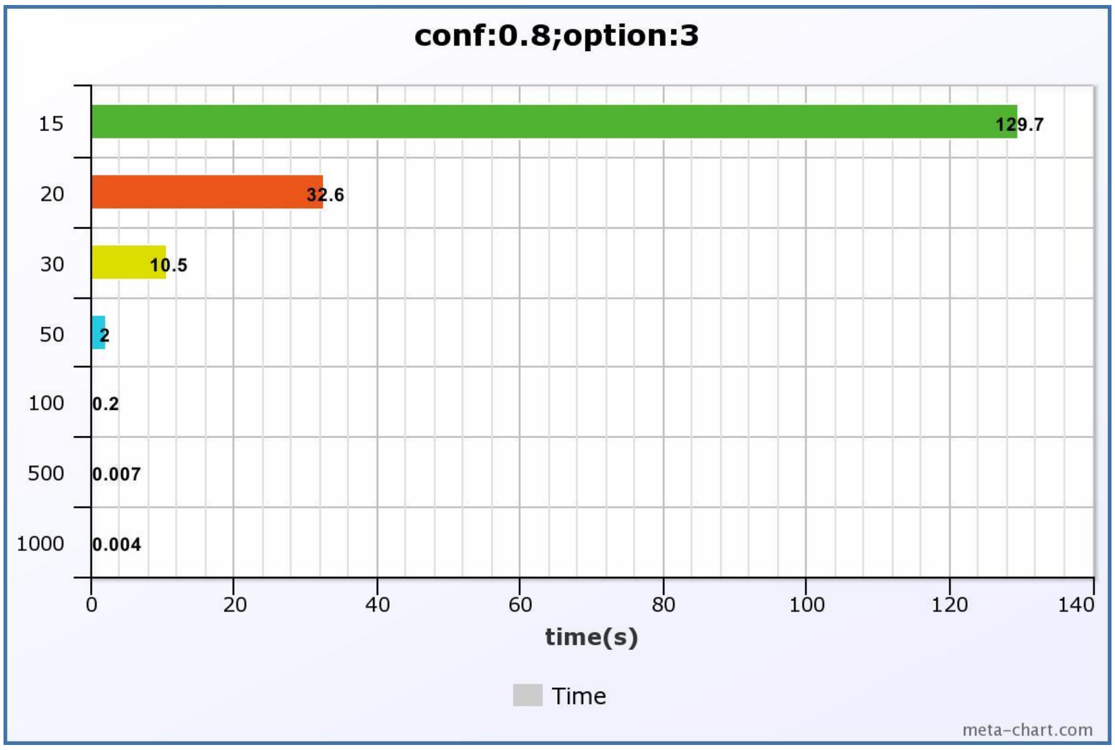
Because of the low efficiency of my code, I don't have the time to run all the combination of minconf and options value using large dataset before due time.

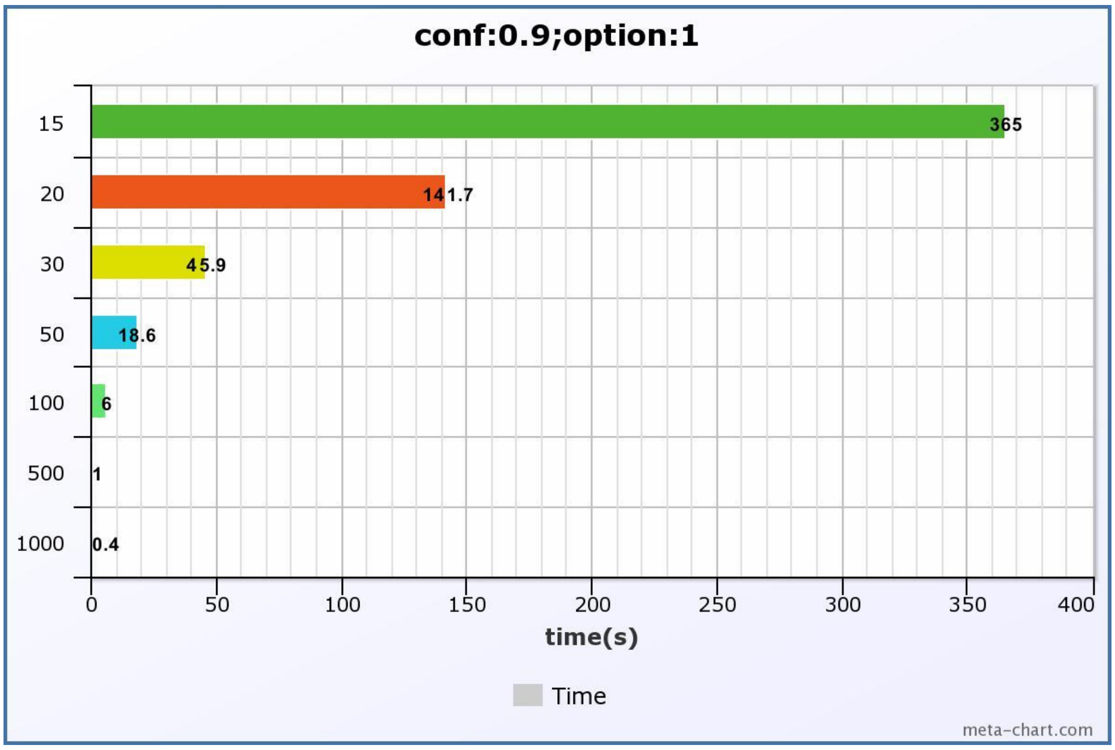
Therefore, the following charts are based on the **small dataset**. I guess it's enough to show the correctness of my code.

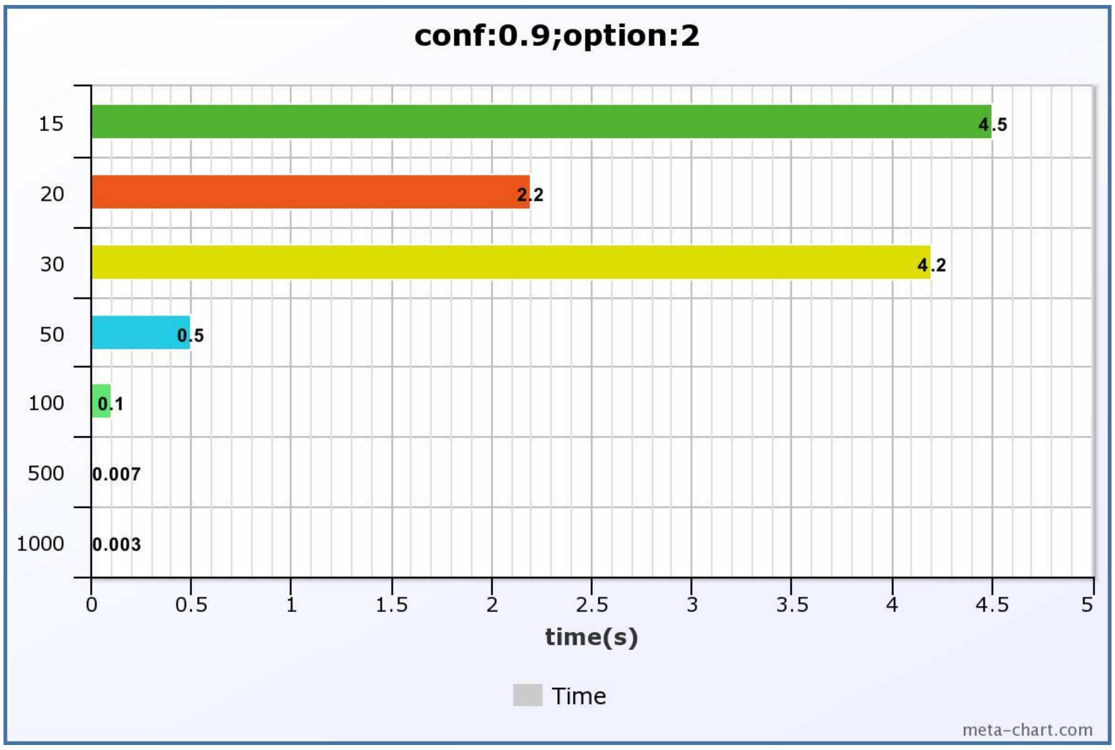
Option 2 results in faster execution time because ordering the items from least support to greatest support ensures that the computationally heavier branches of the enumeration tree have fewer relevant transactions. This helps maximizing the selectivity of projections and thus ensures better efficiency.


conf:0.8;option:1
15
20
30
50
100
500
1000
365
141.7
48
16.9
6.3
0.86
0.48
0
50
100
150
200
250
300
350
400
time(s)
Time
meta-chart.com

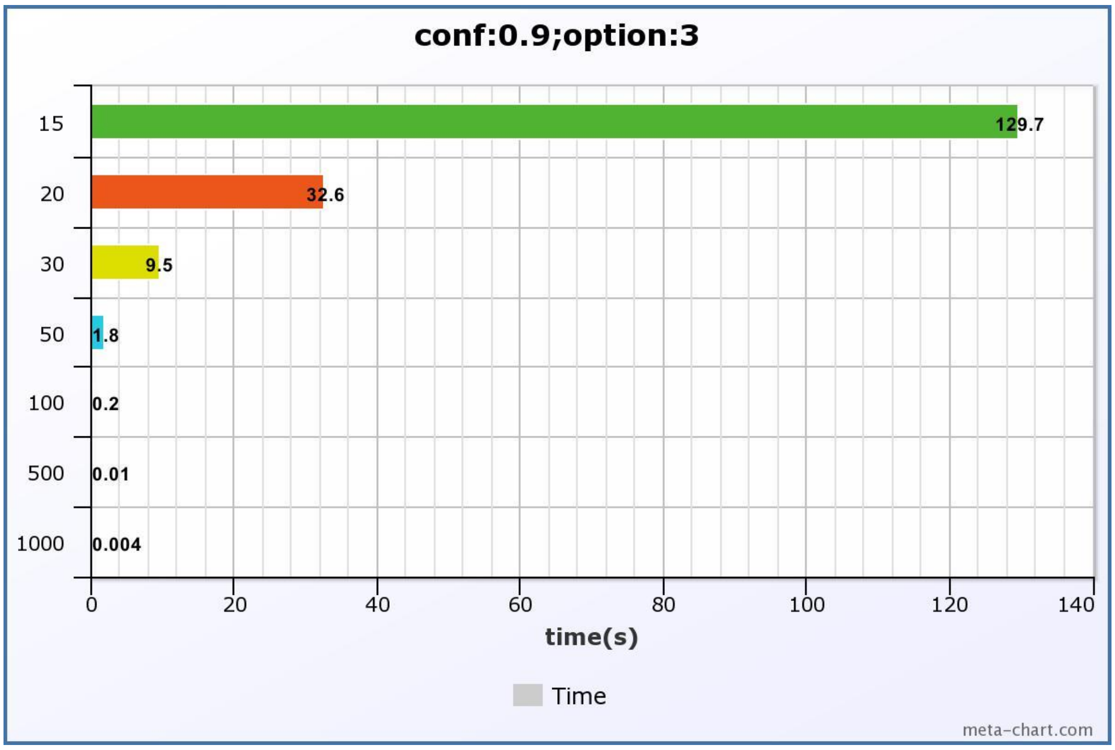

conf:0.8;option:2
15
20
30
50
100
500
1000
4.5
2.2
5.9
0.5
0.1
0.007
0.004
0
1
2
3
4
5
6
7
time(s)
Time
meta-chart.com


conf:0.8;option:3
15
20
30
50
100
500
1000
129.7
32.6
10.5
2
0.2
0.007
0.004
0
20
40
60
80
100
120
140
time(s)
Time
meta-chart.com

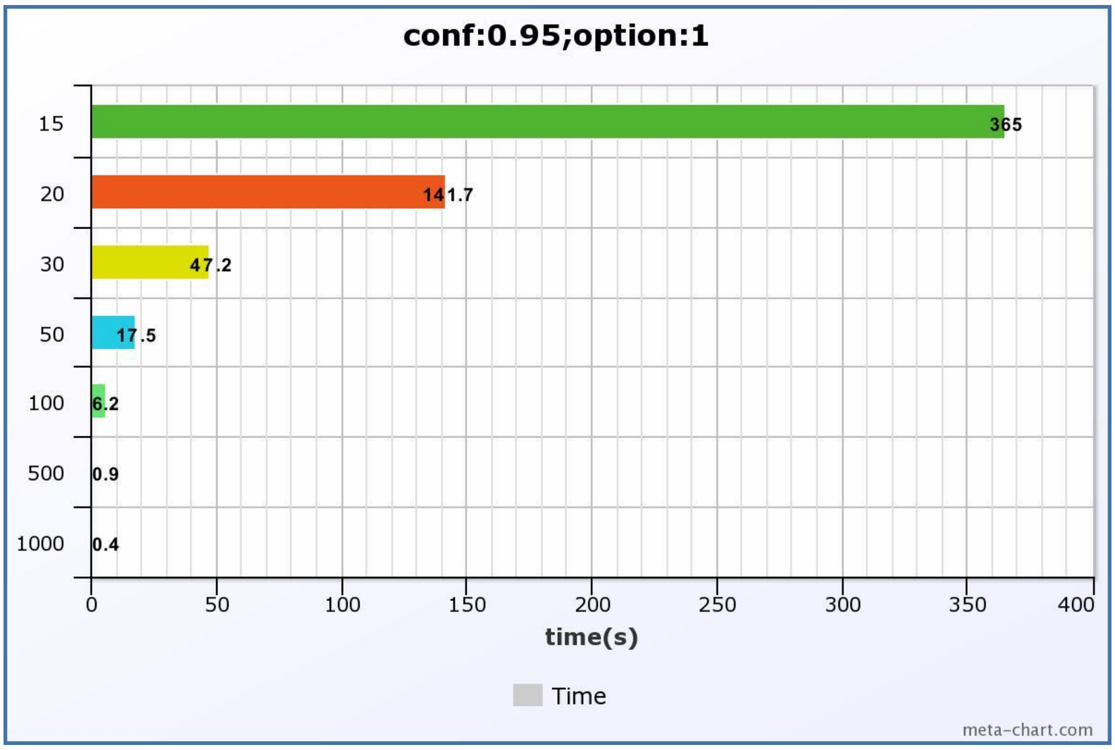

conf:0.9;option:1
15
20
30
50
100
500
1000
365
141.7
45.9
18.6
6
1
0.4
0
50
100
150
200
250
300
350
400
time(s)
Time
meta-chart.com

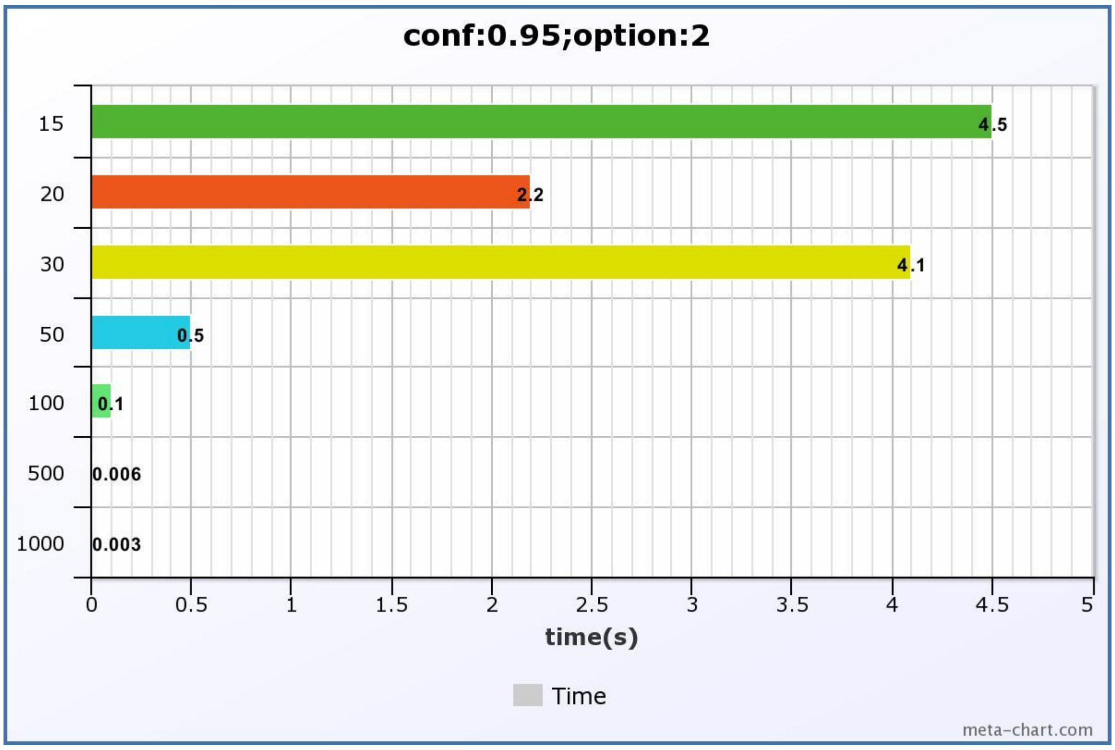


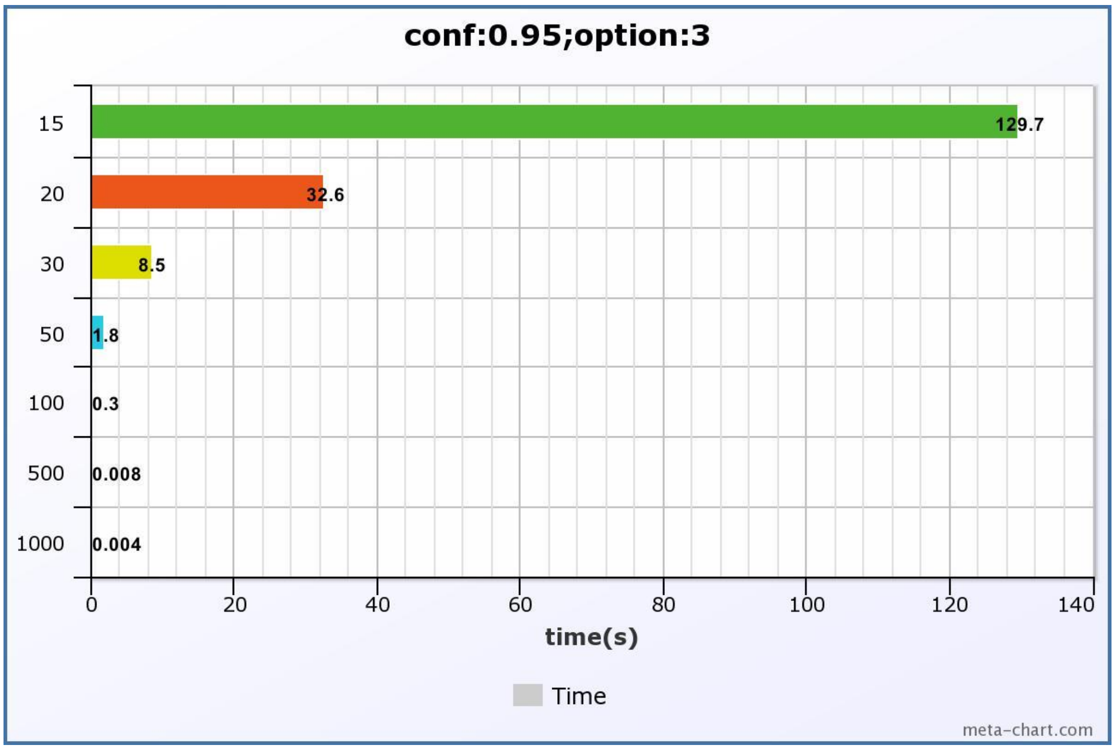
conf:0.9;option:2
15
20
30
50
100
500
1000
4.5
2.2
4.2
0.5
0.1
0.007
0.003
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
time(s)
Time
meta-chart.com

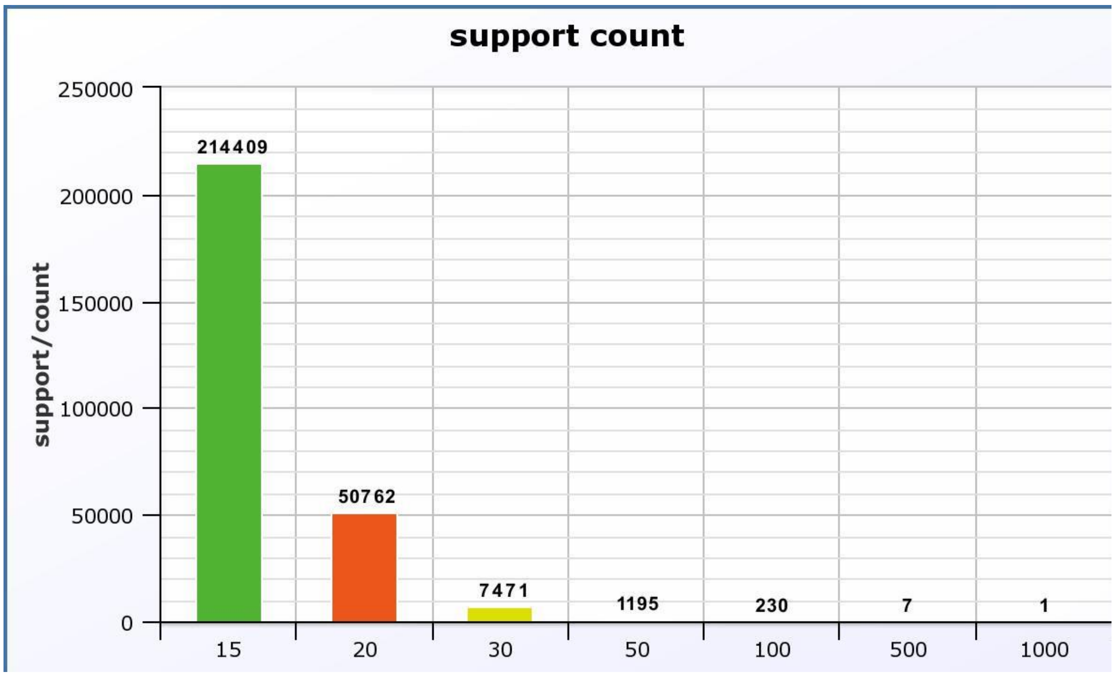


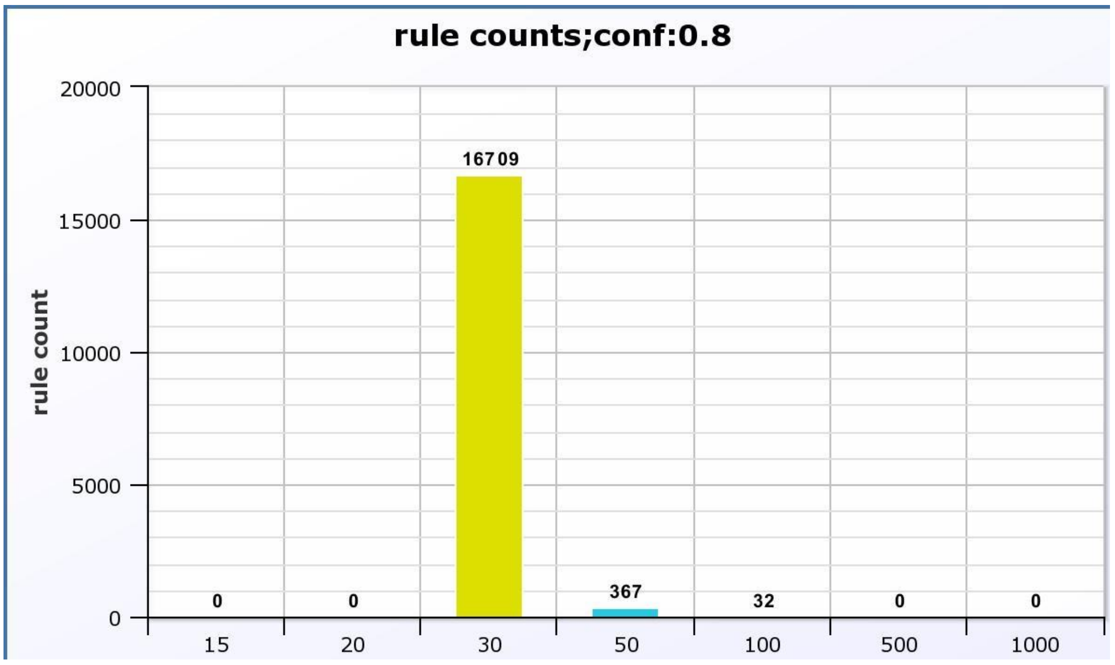
conf:0.9;option:3
15
20
30
50
100
500
1000
129.7
32.6
9.5
1.8
0.2
0.01
0.004
0
20
40
60
80
100
120
140
time(s)
Time
meta-chart.com


conf:0.95;option:1
15
20
30
50
100
500
1000
365
141.7
47.2
17.5
6.2
0.9
0.4
0
50
100
150
200
250
300
350
400
time(s)
Time
meta-chart.com

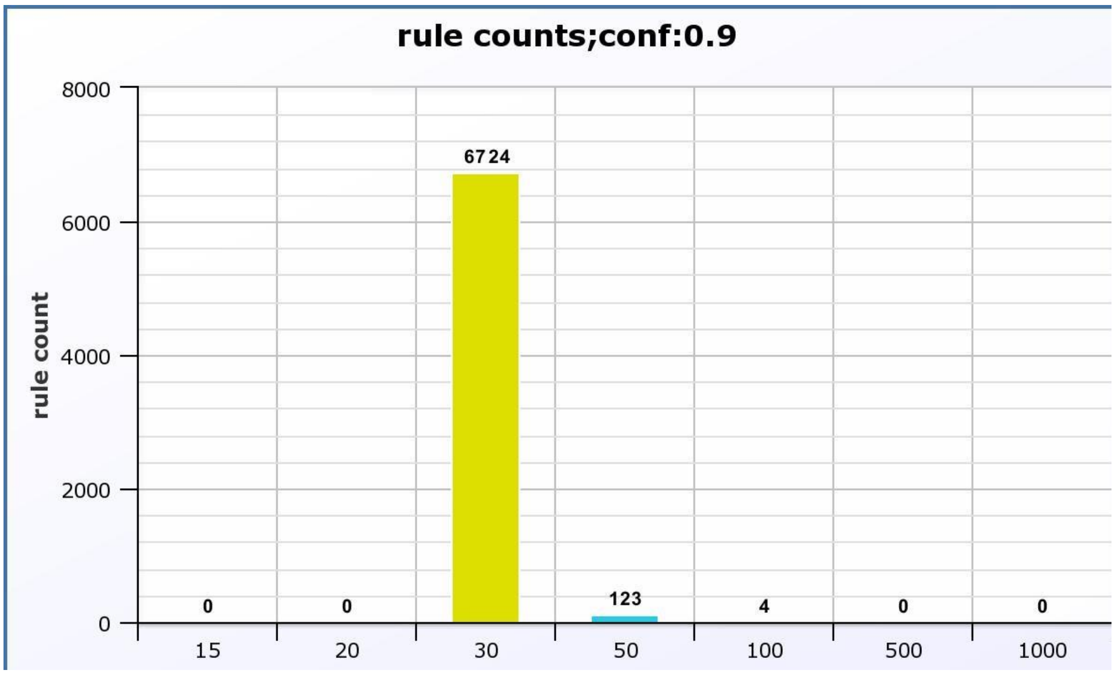

conf:0.95;option:2
15
20
30
50
100
500
1000
4.5
2.2
4.1
0.5
0.1
0.006
0.003
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
time(s)
Time
meta-chart.com


conf:0.95;option:3
15
20
30
50
100
500
1000
129.7
32.6
8.5
1.8
0.3
0.008
0.004
0
20
40
60
80
100
120
140
time(s)
Time
meta-chart.com

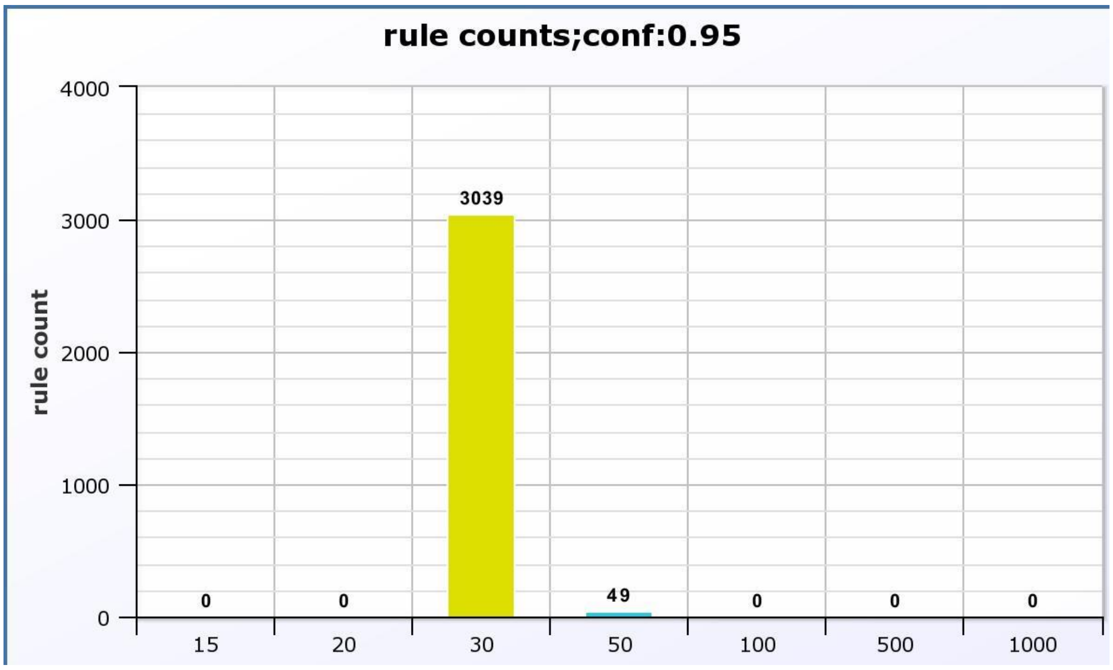

support count
support/count
250000
200000
150000
100000
50000
0
214409
50762
7471
1195
230
7
1
15
20
30
50
100
500
1000



rule counts;conf:0.8
rule count
20000
15000
10000
5000
0
16709
367
32
0
15
20
30
50
100
500
1000



rule counts;conf:0.9
rule count
8000
6000
4000
2000
0
6724
123
4
0
15
20
30
50
100
500
1000


rule counts;conf:0.95
rule count
4000
3000
2000
1000
0
0
0
3039
49
0
0
0
15
20
30
50
100
500
1000