APPROXIMATION OF ICEBERG-CUBES USING DATA REDUCTION TECHNIQUES

THESIS

VITA

Leon Bukhman was born in Gomel, Belaru

AN ABSTRACT

APPROXIMATION OF ICEBERG-CUBES USING DATA REDUCTION TECHNIQUES

by

Leonid Bukhman

Advisor: Hervé Brönnimann

Submitted in Partial Fulf

equivalent size. An iceberg-cube generator is

TABLE OF FIGURES

random sample outliers will minimize the distance function. EASE is able to obtain its sample with a single pass over the data, making it ideal for processing a data stream. This

defined for dealing with association rules are *support* and *confidence*. The support of an itemset `x` in `D` is defined as `frequency(x)/n`

Data cubes can be represented naively as multidimensional arrays. The summary fields are placed in the array by conceiving of a sentinel/wildcard item for each dimension at index zero. To represent a multidimensional array as a one-dimensional array, the index to the one-dimensional array can be computed as the sum of products of indexes times their respective dimension length of all dimensions. Although this implementation provides constant time access to the data, it requires a lot of space. In fact, a d-dimensional cube with n_i distinct items per each dimension i requires $\prod_{i=1}^{d}(n_i+1) = O(n^d)$ counters to be calculated exactly. For sparse cubes, this can be a tremendous waste of memory. An alternative implementation uses a hash table with a hash of the ordered set of items

On-Line Analytical Processing (OLAP) has become a popular technology for analyzing multidimensional data. While there are many vendors offering various products and

requirements providing an even greater benefit than when sampling flat/transactional data.

1.2.3 Iceberg Cubes

When data cubes are used for decision support, all the cells of the cube are not generally used. This is especially true for sparse data cubes that only contain a few cells with values. Because computing and maintaining a complete data cube takes up so much space and time, it is of2(pu7f(u)0.ube r)-64(e,-0.5(sene)fiialland mpractic]TJ19.131520 TD-0.00089Tc0.0289

aggregate (i.e. low-numbered itemset) does no

trees (FP-tree) [10] and hyper trees (H-tree) [14]. However, all these approaches require more memory and time than an algorithm that computes the iceberg cube from a sample of the data. If exact results are not required, data reduction can provide a much more practical alternative for dealing with multidimensional data. Furthermore, data reduction techniques can be applied to streams of data while the algorithms mentioned above require multiple passes over the data.

1.3 Motivation

The various candidates for data reduction discussed in section 1.2 all lend themselves to several applications. The first possible application of an algorithm that can create a sample of a dataset is the use of the data sample to provide a compact representation of a data cube or an iceberg cube. For this to be useful, the sample has to maintain several properties. First and foremost, the amount of space occupied by the sample would have to

) u (7 be considerably smaller than the space occupied by the sample, -2.2946 T107 Tw 0.11 T34 Tw[(7.9(e

Lossy Counting algorithm [5] are used to reduce memory requirements while providing an error bound for the approximation. The goal is to create an algorithm that will generate a samfle that can be used to approximate an iceberg cube of frequency counts generated from the original data. This algorithms must have a small memory footprint, have

where the number of distinct items n is much greater than the number of items with

and the associated elements; those elements are a superset of the items with minimum

the algorithm are not as small as others that provide the equivalent accuracy. More importantly, this algorithm allows *false negatives*. Namely, elements whose frequencies

The maximum number of counters kept at any time by Lossy Counting is bounded by $^{1}/_{ů}\log(ŮN)$

The complete implementation of the Lossy Counting algorithm is provided below as an infinite loop over a stream of data. Note that line 12 can be performed at any time

different items. In addition, because we are dealing with a “count” data cube, all the projections of a heavy cell are also heavy.

Lemma 2.4: In a d-dimensional data cube, the number

sample size of

running time complexity is $O(n \times l)$. As shown in Section 3.3.1, this implementation can be simplified and improved in practice. Although the asymptotic bounds remain

3.2.3 Application to Data Cubes

In order to be useful for generating data cubes, a sampling algorithm needs to guarantee

To guarantee an accurate frequency count at each level of the cube, the sample needs to be constructed with penalty data for each cell of the cube. If we use EASE as the base algorithm for sampling a data cube, then the frequency error at each cell is bounded by ů.

as the order of transactions could influence the transactions that are selected and alter the sample quality. Unfortunately, this method di

value. Approximating the size may not always be practical since, in the case of a stream, all of the data is used rather than a fixed size random sample. Furthermore, the per-item overhead in the algorithm is significant in practice although not reflected in the asymptotic T i9

the transaction is taken. In addition, notice that this value can simply be computed for each item and the sum of all these va

3.3.3 Biased L2 Algorithm

The Biased L2 algorithm also proposed in [1] uses an even simpler and more refined quadratic penalty function to generate a sample. Bein2 14orashoot4or EASE, it similarly trims away transaction to reduce the size4or the sample while maintainin the approximation properties or the original datase2c -0.0016 becos aw(1ion 4.7 -5.3(:)- m)1 7.86

If this condition holds, the transaction is accepted and r_i is incremented.

Refer to [1] for a proof of why it is always possible to pick a transaction such that the penalty function does not increase. The paper also proves that the size of the sample is $|S_r| = \alpha N + O(\sqrt{\alpha(1-\alpha)\times mN})$ and that the discrepancy between the Biased-L2 sample and the actual data is bounded by $\varepsilon = O(\sqrt{((1-\alpha)\times m)/(\alpha N)})$. These properties are

Biased-L2 (D, Ŭ1.-L2 (D2

3.4 Evaluation of Sampling Algorithms

3.4.1 Data Used

The data used for the evaluation of all the algorithms in sections 3 and 4 was computer

$$PR_1(D,S) = \frac{Dist_1(D,SRS)}{Dist_1(D,S)}$$

Note that testing was performed using other distance functions as well (e.g. the Root Mean Square distance). The following formula was used to measure the RMS distance:

$$Dist_2\ D\ S \quad \sqrt{\sum_{A\ I(D)} f\ A\ D \quad f\ A\ S\ ^2}$$

$$length\ 8T\text{-}18\ 5\ 52D$$

For a better picture of the sample's quality in approximating a data cube, the function above was also evaluated separately for each level of the data

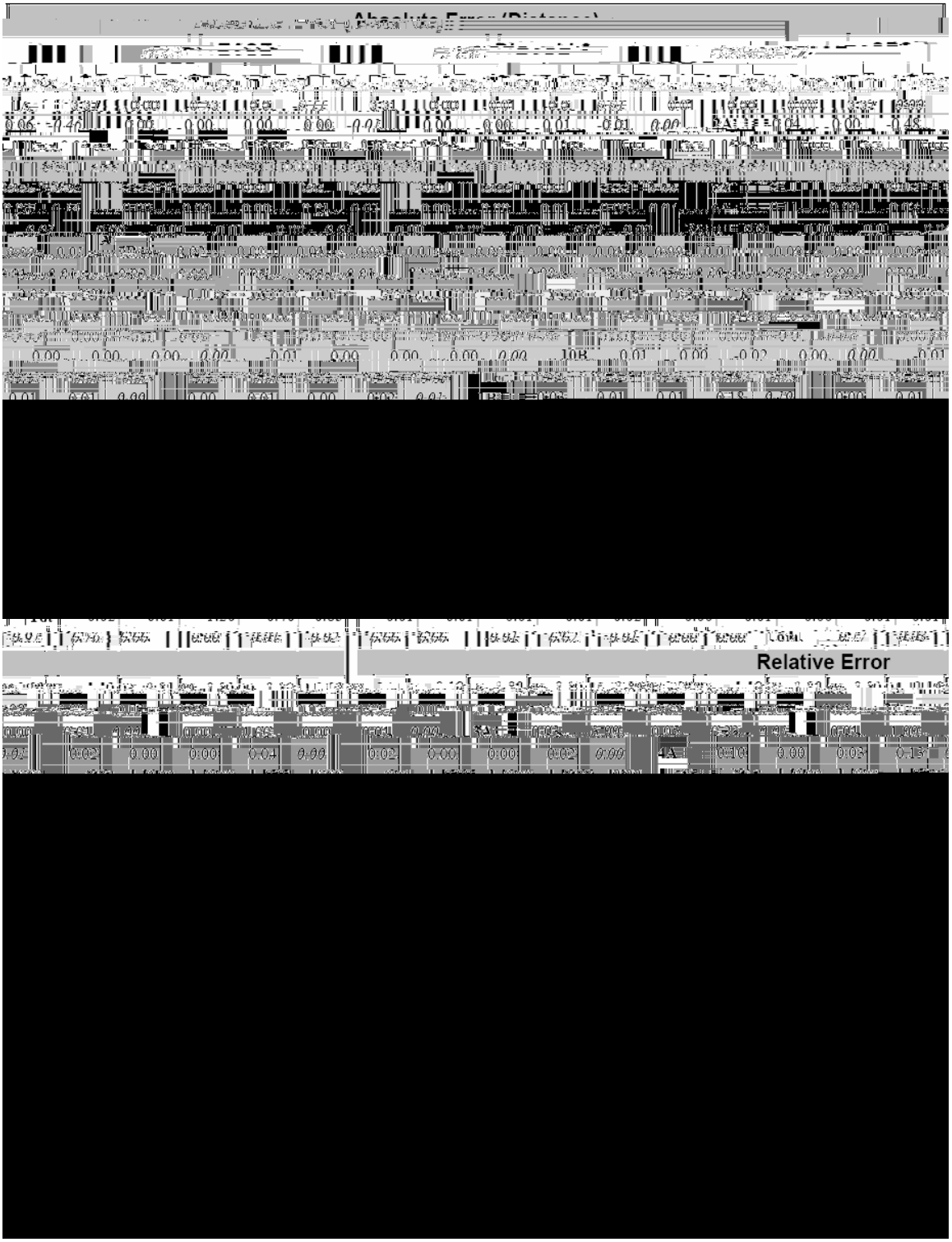
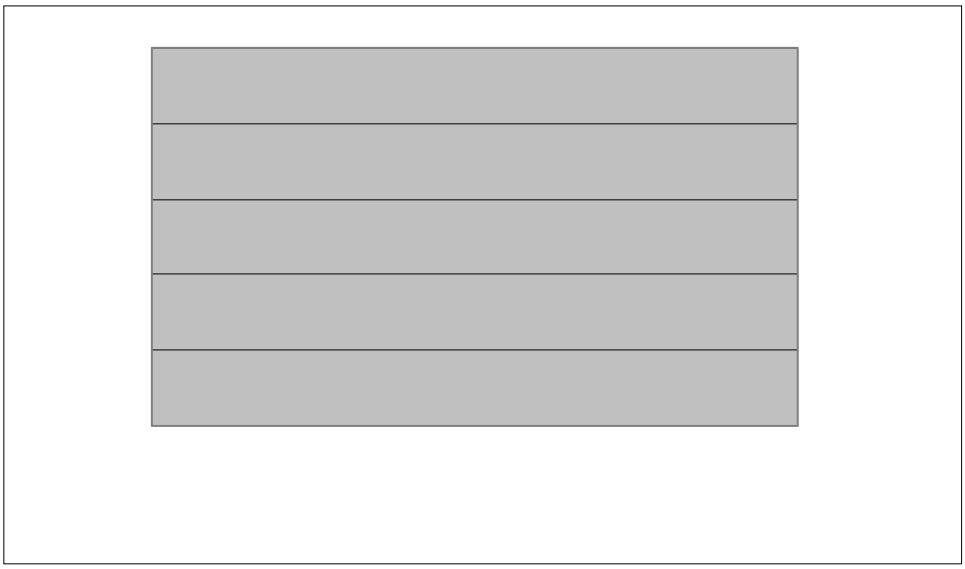


Figure 8 - Table comparing SRS, EASE, and Biased-L2 relative and absolute errors



4. APPROXIMATION OF ICEBERG-CUBES

As stated throughout this thesis, iceberg cubes can provide a practical and efficient way to analyze large quantities of data. Section 3 explained how to maintain a data sample for the purpose of approximating a data cube of the original dataset. Th.408extst-1.3(e4.9(sp1.1(. s

initialize all the penalty data before starting

that dynamically allocate memory as needed while providing constant time access to stored data.)

One difficulty with using EASE is that the penalty information for an itemset is not directly related to its frequencyy aesb. This prevents us from knowing which itemsets are actually frequent without maintaining this information separately. Doing this however, would require an extra array of itemset c

algorithm remains $O(m \times \log n)$

and trivially since this penalty data can also

```
MinSupport_BiasedL2Sample (D,
```

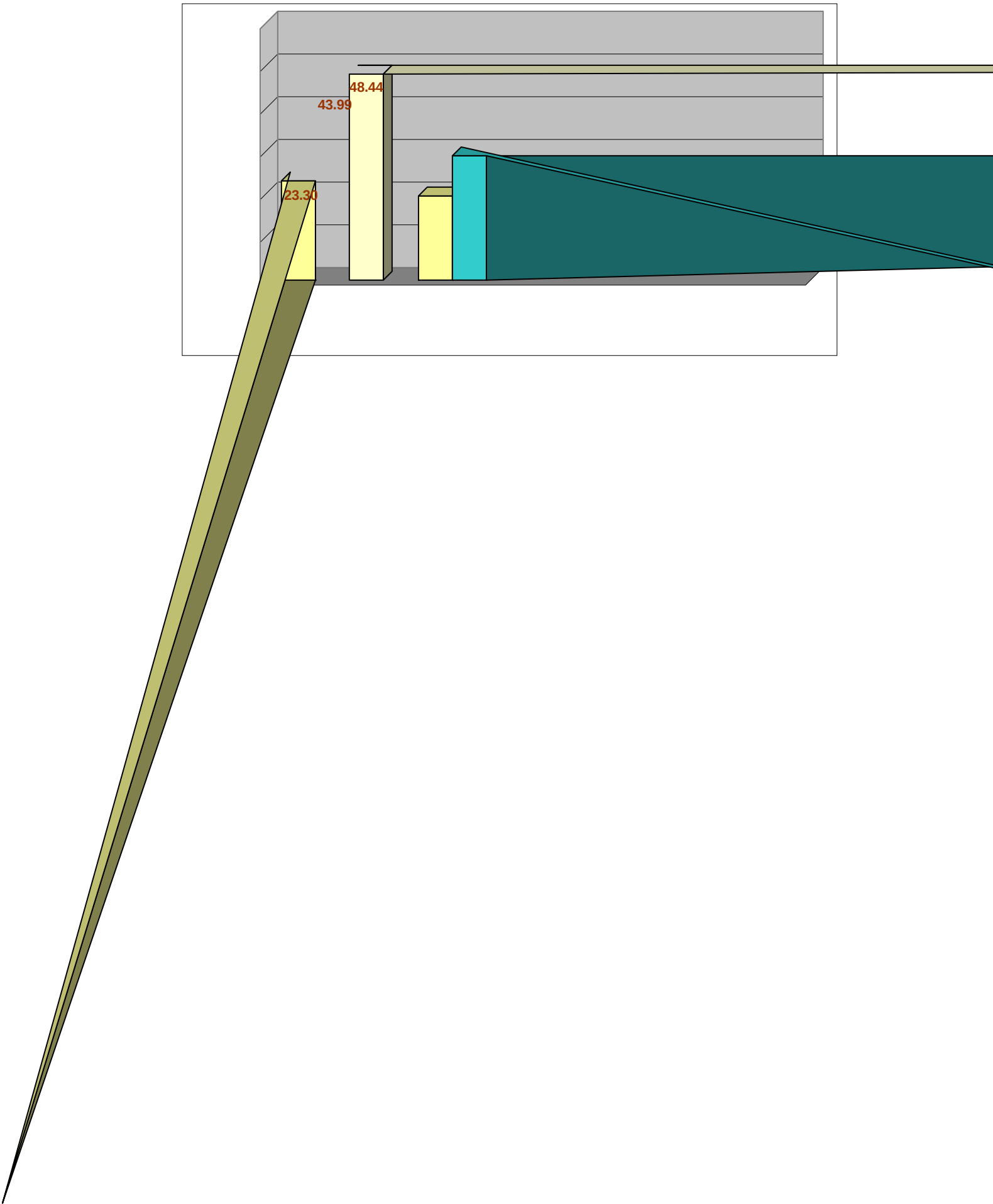
section 3.4.2 was used as the benchmark. The minimum support cutoff value was s=0.05±Ů with an error bound Ů=0.005. In addition, unless otherwise noted, the sampling rate used for evaluating Biased-EA and Biased-L2 is Ŭ=0.03 and EASE is set to undergo 5 rounds of halvings. Th to generate the performance ratio was changed from the one used in section 3.4.3 to identify minimum support itemsets as follows:

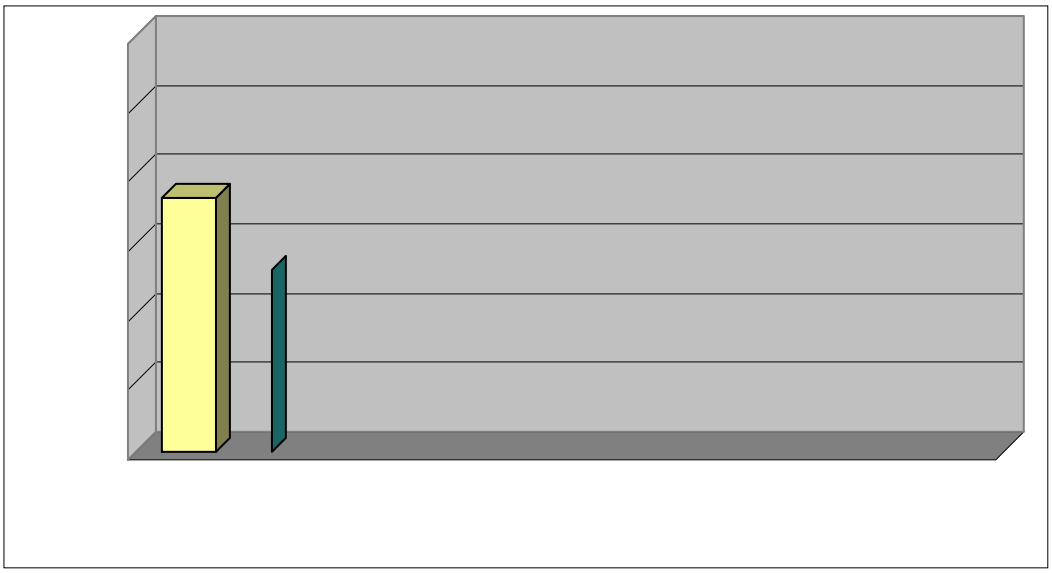
```
SELECT sum(abs(freq-freqRand))/sum(abs(freq-freqDet))
FROM tblStructuresOLAP
WHERE freq>s
```

4.3.2 Comparison of Algorithms

The first obvious evaluation made was to compare thance ratios (over SRS) of the three algorithms. As expected, the results in Figure 14 show that Biased-L2 has a clear lead in overall performance over the other two algoriths. One surprising

that both EASE and Biased-EA outperformed Biased-L2 for sampling the 3-itemsets. This is possible because Biased-L2 is the only one of the algorithms to actually prune away penalty data that could be useful. The implementation of EASE and Biased-EA above does not actually prune the penalty data m infrequent itemshich means that it is maintaining more data in memory.


48.44
43.99
23.30



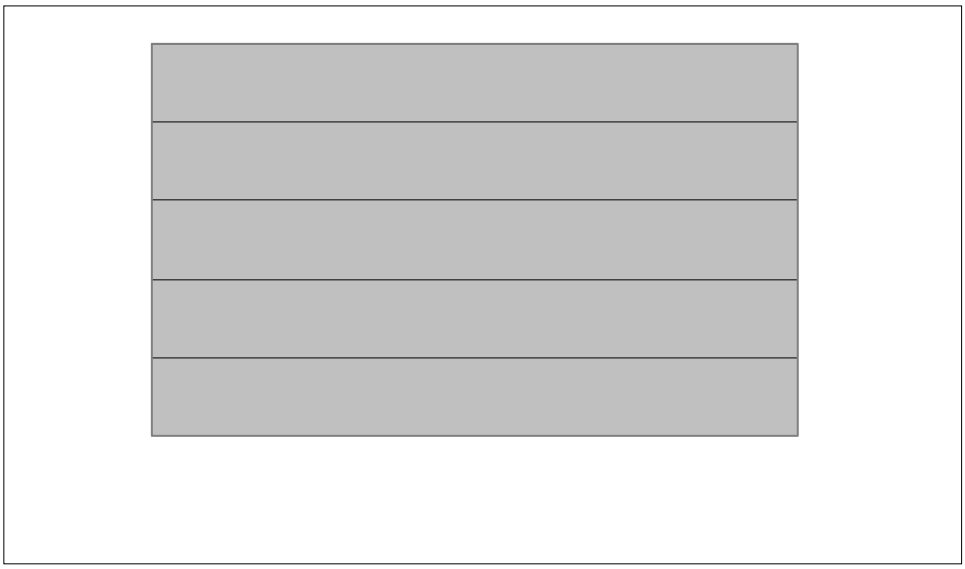
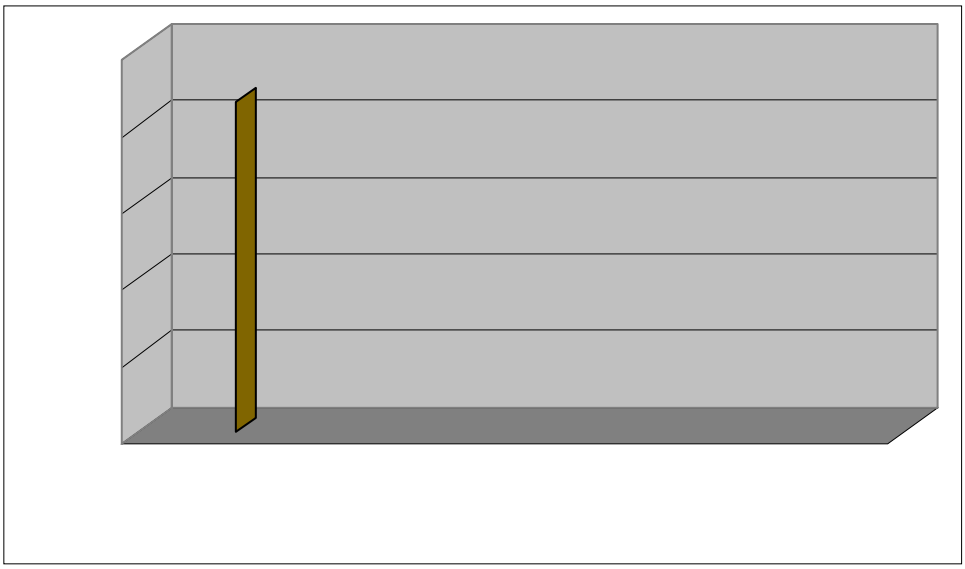


Figure 17 is an iceberg cube of the original data showing cells whose actual frequency is greater than or equal to 5.00%. The table compares the quality of the MinSupport_Biased-L2 algorithm to that of SRS when sampling for itemsets with minimum support. The table is arranged the same way as Figures 6 and 7. Namely, each cell in the table is organized as follows:

Count of original data EASE or L2 percen



memory complexity test were not at all surprising. EASE requir

5. GENERATION OF RANDOM ICEBERG-CUBES

A new test dataset was needed in order to prove that the results in section 3.4.6 and 4.3 are not strongly dependent on the data used

A formal description of the problem at hand is as follows. Given `n` = number of transactions, `numFreq` = number of frequent

```
buildIcebergCubeDataset (n, numFreq, s, noise)
1.   A = {}                        //empty array for random numbers
2.   A[0] := 0; A[numFreq] := 1-noise-s*numFreq
3.   if (1-noise)/numFreq < s then return //sanity check
4.   for (i=1; i<numFreq; i++)    //generate random numbers
5.       A[i] := rand(1-noise-s*numFreq)
6.   Sort A
7.   // generate frequent cells
8.   for (i=0; i<numFreq; i++)
9.       records = floor(n*(s+A[i+1]-A[i])+.5) //round
10.      ri = random itemset with 3 items in range [1,10]
11.      for (j=0; j<records; j++) save ri
12.  s+A[i+1]-A[i])+.5) //round
```

1976101TD0T8443T0/TD11760GTD0 Tc( )Tj/TT11 1 Tf0.8443 0 TD-0.0013

however, to make the algorithms more in line with the dataset used, the min support for

6. CONCLUSION

Three data sampling algorithms were explored in order to produce a sample that can be used to approximate an iceberg cube of the original data. Section 3 shows how the sampling algorithms can be utilized to generated samples used to approximate frequency count data cubes of the original data. The algorithms in section 4 show how Lossy

the sampling itself can run in fractions of a

```
    if $generateEASEMinSupportSample;
generateEAMinSupportSample()   and print "Biased EA sample generated\n"
    if $generateEAMinSupportSample;
generateL2MinSupportSample()   and print "Biased L2 sample generated\n"
    if $generateL2MinSupportSample;
rebuildResultsTable()          and print "Results table rebuilt\n"
    if $rebuildResultsTable;
analyzeRandom()                and print "Random sample analyzed\n"
    if $analyzeRandom;
analyzeSample()                and print "Sample analyzed\n"
    if $analyzeSample;
print "Score was " .           getScore() ."  ".
                               getScore(1)."  ".
                               getScore(2)."  ".
                               getScore(3)."\n Mem Usage: ".$mem."\n"
    if $getScore;

#dTsconnect
$dbh->dTsconnect();

### Main Code
##############################################################################

#define the transaction (given a record)
sub getTransaction {
    my $rec = shift;
    my @items = (0,0,0,0,0,0,0,0,0);

    my $b = $d{1{$rec->{'Boro'}};


        sub generateL2MinSupponsaction {
        $rec}
```

```
#Main loop
while (my $ref = $sth->fetchrow_hashref) {
    $N++;
    $BucketID = int(ceil($e * $N));
    my @items = getTransaction($ref);

    my ($sum_f, $sum_r)=(0,0);
    foreach my $i (@items) {
        $t[$i] = $BucketID if ! $f[$i];
        $f[$i]++;
        $sum_f += $f[$i];
        $sum_r += $r[$i] if $r[$i];
    }

    #keep transaction if
    if ($len,D( ($9m_r += $<[$i]ate*_f +=  )Tj-10.78 -1.1557 TD(         )Tjsum_
```

```
for (my $i=0; $i<$h; $i++) {
    $delta[$i] = sqrt( 1-exp( -log(2*$m)/($n/(2**$i)) ) );
}

#main loop
while (my
```

```
                $Q1[$i] *= (1-$delta)**$rate;
                $Q2[$i] *= (1+$delta)**$rate;
            } elsif ($Q1temp[$i]) {         #in current bucket fnly
                $Q1temp[$i] *= (1-$delta)**$rate;
                $Q2temp[$i] *= (1+$delta)**$rate;
            } else {                        #first time
                $Q1temp[$i] = (1-$delta)**$rate;
                $Q2temp[$i] = (1+$delta)**$rate;
            }
        }

        my $sum = 0;
        ffreach my $i (@items) {
            if ($Q1[$i]) {                    #if frequent
                $sum += ($Q1[$i]-$Q2[$i]);
            } elsif ($Q1temp[$i]) {           #in current bucket
                $sum += ($Q1temp[$i]-$Q2temp[$i]);
            }
        }

        if ($sum<=0) {   #keep it
            ffreach my $i (@items) {
                if ($Q1[$i]) {                    #if frequent
                    $Q1[$i] *= (1+$delta);
                    $Q2[$i] *= (1-$delta);
                } elsif ($Q1temp[$i]) {           #in current bucket
                    $Q1temp[$i] *= (1+$delta);
                    $Q2temp[$i] *= (1-$delta);
                0 -1.1317 TD(         )Tj0    }
"UPDATE tblStructures SET Sample=1 ".

            push (@keptItems, [@items]);
        }


        if ($e*$N == ceil($e*$N)) {
            ffr my $i (0 .. $#keptItems) {   #ffr each kept transaction
                my $aref = $keptItems[$i];
                ffr my $j (0 .. $#{$aref}) {     #ffr each item
                    my $item = $aref->[$j];
                    $Q1[$item] = $Q1temp[$item] if ! $Q1[$item];
                    $Q2[$item] = $Q2temp[$item] if ! $Q2[$item];
                0 -1.1317 TD(         )Tj0    }

            #tItTjTTf058617 TD(                47b 47b 47b /T40i/TT24t1e    47b 470
```

```
  if ($ord) {
      $filter .= "and Ord = $ord";
  }  ($ord) {
$filter .= "and Ord = $ord";
```

```
        my $sql ="insert into tblRandom(D1,D2,D3,Sample,Random,Sort) ".
                 "values ($d1,$d2,$d3,0,0,rand())";
        for (my $j=0; $j<$records; $j++) {
            $dbh->do($sql);
        }
        $transactionsGenerated = $transactionsGenerated + $records;
    }

    return 1;
}
```

9. REFERENCES

1. A. Astashyn. *Deterministic Data Reduction Methods for Transactional Data Sets*. Masters Thesis. Polytechnic University, 2004.

2. H. Brönnimann, B. Chen, M. Dash, P. Haas, and P. Scheuermann. Efficient Data

10. J. Han, J. Pei, and Y. Yin. Mining Frequent Patterns without Candidate Generation. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, 2000.

11. R. Agrawal, T. Imielinski, and A. Swami.