Database Summarization

Rajalakshmi Nandakumar and Laurel Orr
CSE 544 Winter 2015

1. ABSTRACT

Although "Big Data" has been the popular research topic for a few years, the problem of how to manage it has yet to be solved. A common problem data scientists face is that they lack the ability to quickly explore and probe massive datasets. To help solve and better understand this problem, we propose and evaluate two data summarization techniques: sampling and probabilistic summarization. We further compare these methods to determine which situations are best suited to which methods.

2. INTRODUCTION

In today's data-crazed world, the ability to quickly and easily understand and learn from data is of crucial importance. A critical first step in this process is data exploration and ad-hoc processing, where a data scientist begins to probe and summarize the data. The goal of this process is to get a general sense of the data's characteristics and relevancy as well as to begin to narrow down the parts of the data to dive more deeply into. As data exploration is improvised with no set queries in mind, this process needs to be very interactive with fast response times. This is were the challenge begins.

As datasets become larger, standard DMS's are incapable of providing the necessary query interactivity. These systems are not geared towards exploration as they guarantee query correctness and will need to process the entire dataset to answer a query. The traditional way of tackling slow runtime is to re-think and re-tune the system by adding indices, creating better statistics, or adding more compute nodes to create a cluster. But what if this problem can be solved by keeping the system constant and re-thinking the data?

The key insight is that data analysts are willing to sacrifice query accuracy for speed. It is ok if the query result is not exactly correct, as long as it's a good representation. So, instead of the system querying the entire dataset, if the queries are run on smaller summaries, the results would still be close to correct but much faster.

In this paper, we discuss two main methods to transform a dataset into a compact, summarized dataset that is queryable. The first is the standard technique of sampling, and in particular, we will implement uniform sampling and stratified sampling. The second main method is a probabilistic approach that aims to find a distribution that best represents the underlying data.

The key challenge in this process is to provide a summarization that is a close enough representation of the actual data to enable a fast query execution without sacrificing too much correctness. To evaluate the methods, we chose a dataset and three representative data exploration queries and compare the methods' summarized results to the true results using the Earth Mover's Distance [9]. The rest of this paper first reviews the dataset in Section 2.1 and then discusses the three techniques in Section 3, the evaluation queries and results in Section 4, the related work in Section 5, and gives future work and concluding remarks in Sections 6 and 7.

2.1 Dataset

Our dataset is of all flights in the United States from January 2013 to November 2014 taken from [1]. The raw dataset is one table, called Flights, with the attributes date (including time), tail number (physical plane id), origin, destination, the scheduled flight arrival and departure time, the actual flight arrival and departure time, and a break down of the flight time (time spent taxing, in the air, at the gate, etc). There are a total of 11,712,110 rows where each row represents a single flight for a particular day. Although we did our summarization experiments on the raw table to more accurately mimic what a data analyst would do after uploading the data, we did create a normalized flight database that eliminated much of the repetition in the raw table. The normalization led to three main relations

1. Airport(airport_id, city, state)
2. ScheduledFlight(flight_id, carrier, flight_number, origin_airport_id, dest_airport_id, expected_departure_time, expected_arrival_time)
3. FlightInstance(date, flight_id, actual_dept_time, actual_arrival_time)

For the Flight table, the flight number and the origin airport id together are a candidate key. This is because the flights were given the same id for both their to and from journeys; e.g., the flight number was 229 for both Seattle to LA and LA to Seattle. In order to make it simpler to access, we created the primary key of flight id, which is unique.

3. APPROACH

To achieve our goal, we took two different approaches to generate a query-able summary. The first approach is to use sampling where we implement two different methods to sample a database, namely random sampling and stratified sampling. The second approach is a probabilistic approach where we calculate a probability distribution that best fits characteristics of the original database. In the following sections, we discuss these approaches in detail and compare their performance on a sample dataset.

3.1 Sampling

One of the most common and intuitive approaches for faster, query-able databases is sampling. The idea is to choose a subset of tuples from the database to execute the queries on. Since the queries will now operate on a smaller dataset, the execution times are faster. Applying the same principle, we can construct a sampled database that effectively summarizes the original dataset. But the key question here is, what is the best possible method to sample a database? In this work, we considered two different methods to sample the database

3.1.1 Random Sampling

The simplest method is to randomly sample the database. In this method, we randomly choose N tuples to represent the database. To do this, we first generate a random number from a uniform distribution for each tuple and sort the tuples based on this number. Then, the first N tuples in the sorted set is the new sampled database. An example query is shown below using Postgresql's random() function.

```
SELECT *
FROM Flights
ORDER BY random()
LIMIT N;
```

One of the key parameters here is the value of N. Higher values of N ensure higher accuracy of results but does not give as much speed-up in the execution of the queries. Basically, there is a trade-off between the execution time and the accuracy. The best method to choose N is to try different values for N and run common queries on these sampled databases. The smallest value of N that has a good enough accuracy level is then chosen.

Typically, random sampling works well on datasets that are repetitive with less deviation and fewer anomalies. In this case, aggregate queries, like average and count work, efficiently with high accuracy. However, for some datasets, a randomly sampled database might not represent the actual database well. For example, if a value x occurs less often for an attribute, then the probability that a tuple with the value of x for that attribute will appear in the sampled dataset is very low. This means that only limited queries can be run on the samples and still produce good accuracy.

3.1.2 Stratified Sampling

As we saw, simple random sampling will not work for all datasets as it does not include all possible attribute values in it. To avoid this issue, the sampled database should include each unique tuple. But in that case, the sampled database will be of the same size as the original and there will be no gain in the execution times. Instead of choosing these extreme cases, a smarter way to sample would be to include tuples that have unique values for a certain attribute set. This method is what we call stratified sampling. The attribute sets usually include the attributes that are frequently queried and we call these the Query Column Sets (QCS). For example, consider the a flight dataset discussed in 2.1. A typical query on this database would be to list all the flights that departs from a city x and arrives in another city y. It would be of the form

```
SELECT id, flight_name
FROM Flight
WHERE origin_airport = x
AND destination_airport = Y
```

In this case, a good summary should include at least one tuple for each `[origin_airport, destination_airport]` combination that exists in the original dataset. Then this pair `[origin_airport, destination_airport]` is considered a QCS. We now construct a sampled dataset that includes N records for all existing values of this QCS in the original data. Similarly, we construct different sampled views of the database for each QCS. As previous work has shown [4], it is possible to predict these QCS easily, and the set of sampled views will now summarize the dataset.

There are two key parameters to consider in this approach, which we will discuss now. First, is the value of N. For each unique value of the QCS in the original dataset, we could sample the tuples with that value in different ways. One option is to do uniform random sampling. In this case, we will randomly choose a fixed number of tuples (say 10) for each unique value of the QCS. However, this does not represent the actual distribution of each value in the original database. To avoid this, instead of choosing fixed number of tuples, we choose N% of tuples for each value of the QCS. Similar to the random sampling method, the value of N is chosen by experiments with common queries.

The second parameter is the size of the QCS set.The best option would be to identify all QCS and construct a sampled view for each QCS value. However, it takes huge additional space on disk. A better trade off would be to choose QCS that cover a maximum number of attributes. For example, the view constructed for the QCS `[orgin_airport, destination_airport]` can also be used for queries involving only `[origin_airport]`. The reverse is also possible; however, it would cause a small degradation in the accuracy of the result.

3.2 Probabilistic Approach

This approach is to generate a probabilistic representation of our database [10]. The theory behind generating a probabilistic database is based on the Principle of Maximum Entropy. According to Suciu and Ré [8], the probability distribution that best represents a database is the one derived from the MaxEnt Principle. Following the MaxEnt principle, if we let Tup be the set of all possible tuples, then for any relation $R \subseteq Tup$ in our distribution, the probability of R, $P(R)$, is equivalent to

$$Z^{-1} \prod_{\phi \in \Phi} \alpha_{\phi}^{|\sigma_{\phi(R)}|}.$$

Here, Z is a normalizing constant, each ϕ is a selection predicate, such as $[A = a_1]$, and the α's are what we need to solve for. For this project, we will only allow predicates

to be equality selection predicates. We will use all single attribute selection predicates (single attribute histograms) and some unions of single attribute selection predicates (multi-dimensional histograms). For each predicate that we use, we will calculate the size of each predicate (the selectivity), and using these selectivity values to solve for the α's.

In particular, to solve for the α's, we will minimize the squared error between the true selectivity and the expected selectivity of each predicate. The equation we will minimize is

$$\sum_{\phi \in \Phi} \left(\left(\sum_{R \in PWD} |\sigma_{\phi(R)}| * P(R) \right) - |\sigma_{\phi(R_{true})}| \right)^2$$

where PWD is the set of all possible worlds and R_{true} is our original, ground truth database. To solve this equation, we use SciPy's `leastsq` solver [2]. To keep the calculations more simple, we add the constraint that Z, the normalization constant, is equal to one; i.e., add a $(1 - Z)^2$ term in our minimization. This is not required, but it acts as a regularization constraint in our minimization.

In terms of storage, once we have calculated the α's, we store the α's in separate tables, one for each selection predicate. Each table has an attribute (column) for each single attribute predicate used (recall a selection predicate can consist of multiple single attribute selection predicates) and an attribute storing the numerical value of α.

Once we have the α's, we can use the first equation and the fact that the sum of $P(R)$ for all possible relations R is equal to one to calculate the probability of each possible tuple actually existing in the database. We can also use the α's to calculate likelihoods of tuples be in queries without needing to store the original database.

4. EVALUATION

To evaluate all our methods, we compared runtime and summarized relation size to EMD for three queries for each of the three summarization methods. We chose to summarize the raw dataset as it better represents the large-scale, un-normalized data analysts will likely handle. When analysts first get data for exploration, they are going to explore the data before even considering normalization.

Additionally, in order to accurately compute the query execution time, we first cleared the operating system buffer cache. We then restarted the Postgres server in order to clear the database's buffer pool. We then used Postgres' `EXPLAIN ANALYZE` query to gather runtime. This command executes the query but does not return any results and therefore does not time how long it takes to transfer the results to the user. Although in some versions of Postgres this command returns the planning time in addition to the execution time, we only included the execution time as that is reported in all versions of Postgres. Each query was run three times on each of the sampled databases and the average of the three executions is used.

Lastly, to compute the size of each relation, we used the Postgres function `pg_column_size`, which computes the size in bytes of an individual cell. For the sampling techniques, we summed the size of the columns `origin_state`, `dest_state`, and `actual_elapsed_time`, while for the probabilistic technique, we summed the size of the tables storing the α's. We chose this method over the page size because this was a more accurate representation of the raw data and number of rows.

4.1 Queries

The three evaluation queries are shown below.

```
SELECT origin_state, COUNT(*)
FROM Flights
GROUP BY origin_state

SELECT origin_state, dest_state, COUNT(*)
FROM Flights
GROUP BY origin_state, dest_state
```

and

```
SELECT origin_state, AVG(actual_elapsed_time)
FROM Flights
GROUP BY origin_state
```

We chose the first two queries because they represent a very typical type of exploration query, selecting some attribute values followed by a count. This also represents a query selecting the distinct values. Although we could have used more selection attributes, we felt the resulting trend from one and two attributes would be similar for more.

We chose the last query because other aggregate values are common in exploration, such as average. We are using this query to show what results might be like for some non-counting aggregating queries. Although these queries are not exhaustive, we believe they are a good initial set of simple exploratory queries an analyst might run.

One last thing to note is that on the original Flights relation, the execution time of each of these queries is approximately 15 seconds.

4.2 Earth Mover's Distance

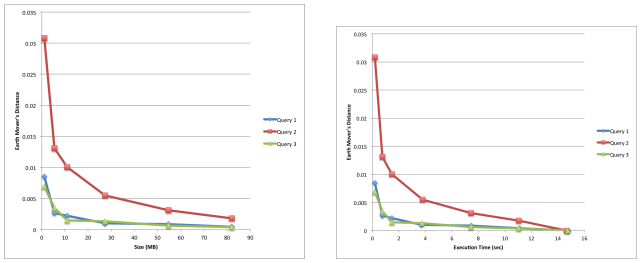
To evaluate the correctness of our query results, we originally thought to use the percent error. So, for each result tuple, where the value is the aggregate, we would calculate

$$\left| \frac{estimated\_value - true\_value}{true\_value} \right| * 100$$

The problem with this formula is that it biases the summaries towards ones that have the smallest difference with respect to the true value. For example, the percent error for a true value of 1,000,000 and estimated value of 1,001,000 is 0.1 percent. The percent error for a true value of 1 and estimated value of 1,001 is 100,000 percent. Although the absolute difference is the same, the second estimated value is deemed a much worse prediction. We do not want this bias in our error because a summarization should produce a globally close result, not just focused on those with small true values.

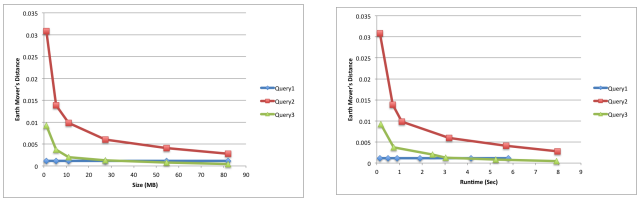
Even if we could decide upon a metric that looked at the numerical difference in values, we also would need a way to evaluate the correctness of queries without aggregates. Although all of our sample queries have an aggregate value, it is future work to include those without. So, some solely numerical error will not suffice. This is why we turn to the Earth Mover's Distance (EMD) [9].

The EMD is a way of measuring the difference between two multi-dimensional histograms. If you think of two distributions of data as two distributions of piles of earth (each histogram bin is one pile), the EMD calculates the minimum work (earth $\times$ distance) to transform one pile of dirt into the other (for discrete distributions, like ours, this is actually just an instance of the transportation problem). Before



(a) Size in MB vs EMD (b) Runtime versus EMD

Figure 1: Evaluation of random rampling technique



(a) Size in MB vs EMD (b) Runtime versus EMD

Figure 2: Evaluation of the stratified sampling technique

computation, you normalize the piles to make sure they each have the same amount of initial dirt, and to calculate the distance, any distance metric can be used to give the distance between any two histogram bins.

For our queries, each bin is a tuple in the `GROUP BY` clause. So for the first query, the histogram is one dimensional, and we have one bin for each origin state. The amount of earth is equivalent to the normalized `COUNT(*)`. The second query is similar, and for the third query, we again have one bin for each origin state, and the amount of earth is equivalent to the normalized `AVG(actual_elapsed_time)`.

An alternative set up is to make each bin be all attributes in the result tuple, including the aggregate value, and make the amount of earth in each bin be uniformly distributed. Although this should have produced an equivalent distance ordering (assuming the aggregate values were scaled to be from 0 to 1), the distribution size (number of bins) is much larger in this method, and when we ran our EMD solver [3], the EMD calculation runtime became too expensive to be reasonable. Future work is to truly investigate this alternative formulation to show it is equivalent.

This distance function we used is the euclidean distance between each attribute. If the attribute is numeric, the distance is the difference in value. If the attribute is textual, the distance is 0 if the values are the same and 1 otherwise. For example, the distance between `'CA'`, `'CA'`, 0.6 and `'CA'`, `'TX'`, 0.1 is

$$\sqrt{0^2 + 1^2 + 0.5^2} = 1.12.$$

By using the EMD as our metric, we are more accurately able to determine how close our result tuples are to the true tuples without needing aggregate values in our queries.

4.3 Results

4.3.1 *Random Sampling*

To evaluate this method, we first constructed multiple summarized versions of the Flights table by varying the value of N. We chose 6 values for N, namely 1%, 5%, 10%, 25%, 50%, and 75% of the total raw dataset size. The processing time required to construct each of these views was approximately 1 to 2 minutes. We then ran the three queries on each of these sampled views.

Fig. 1a plots the EMD for the three queries versus the size of the sampled view. As a reference point, the EMD distance when assuming a uniform distribution for query 1 is 0.456, query 2 is 0.584, and query 3 is 0.113 (these are the same for the stratified sampling). One thing to note when calculating the uniform distances is that we only used the tuples that appeared in the true result set, not all possible tuples. This becomes important when considering the probabilistic results explained below.

We can see that the EMD decreases for all the queries as the size of the sampled database increases, and this is expected as the more data you have, the closer the result is to the truth. We also notice that the EMD is less than 0.01 for query 1 and query 3 and less than 0.03 for query 2, even if we use a 1% randomly sampled database of size 1 MB compared to the actual database of the size 109 MB. This is due to the fact that the state attributes in the tuples in the raw Flights table were repeated. As there are only 53 × 53 (2,809) possible origin and destination state pairs but over 11 million rows, each row is duplicated an average of roughly 4,000 times. This is because the flight schedules rarely change and each flight occurs an almost equal number of times over the two years in the dataset.

Finally Fig. 1b plots the EMD against the execution time of these queries in each of the sampled databases. We can see that as the size of the database increases, the execution time increases and correspondingly the EMD decreases. In this case, all the three queries can be run for less than 1 second compared to the 15 seconds it takes to run on the original dataset.

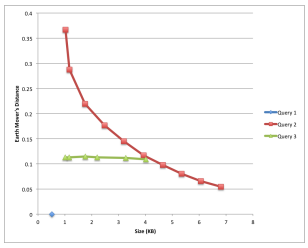
4.4 Stratified Sampling

To evaluate the stratified sampling method, we first defined the QCS as `[origin_state]`. We then did a stratified sampling on the raw dataset and produced multiple views by differing the value of N. We similarly chose 6 values for N namely 1%, 5%, 10%, 25%, 50% and 75% of tuples for each unique value of the QCS. The processing time required to construct each of these views was approximately 2 minutes. We then ran the three queries on these relations.

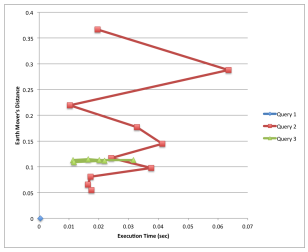
Fig. 2a plots the EMD for the three queries against the size of the sampled relations, and Fig. 2b plots the EMD for the three queries versus the execution time of these queries on the sampled relations. If you notice, for the second query, the actual QCS should have been `[origin_state, destination_state]`. However, we still ran it on the `[origin_state]` QCS sampled relation to observe the error caused by using a small QCS.

As we already saw, since the attributes used in these queries were highly repeated, there was not a huge difference between the random sampling and the stratified sampling technique for this data. In general, we saw good accuracies for these queries on a 1% sampled relation with an execution time of less than 1 second compared to 15 seconds it takes on the original relation.

However, there are a couple of key things to notice. For query 1 using stratified sampling, where we measure the count of the flights for each origin state, the EMD will always be constant and close to 0 irrespective of the size of



(a) Size in KB versus EMD



(b) Runtime versus EMD

Figure 3: Evaluation of the probabilistic summarization technique

the sampled relation in contrast to the random sampling technique. This is because in stratified sampling, the tuples were sampled proportional to the actual number of tuples for each unique value of the QCS.This means that the sampled database actually represents the relative count for each unique value of the QCS. For example, for N of 1% and a relation where there are 1,000 tuples satisfying `origin_state = 'TX'`, the sample relation will have 1% of 1,000 sample tuples where `origin_state = 'TX'`. For an N of 5%, the sample will have 5% of 1,000 tuples. Therefore, the proportional amount of tuples in each QCS remains constant for each N.

Another aspect to note is that if there was a query with the some selective value as a predicate, such as airport id and date, then the stratified sampling technique would work better than the random sampling technique because the QCS sampled relation will include all possible predicate values, even though those values do not occur very frequently in the entire relation. The random sampling technique would likely not have tuples satisfying the predicate on these attributes.

4.4.1 Probabilistic Summarization

To evaluate the probabilistic summarization technique, we first had to decide how to vary our samples. As the only data we store for this technique is the computed α values, the only way to vary the size was to change the number of variables in our equation. Thus, for the smallest sample size, we used only single attribute histogram values for each attribute in each query. We then added double attribute histogram values (only applicable for query 2 and query 3) from taking the N pairs that contributed most to total number of rows. In other words, we took the pairs that resulted from

```
SELECT attr1, attr2
FROM Flights
GROUP BY attr1, attr2
ORDER BY COUNT(*) DESC
LIMIT N
```

where we varied N as 10, 50, 100, 150, 200, 250, 300, 350, and 400.

The major bottleneck we ran into was processing time. For query 2, an N of 400 took around 4.5 hours for the solver to output a solution. For query 3, an N of 250 took the solver 9 hours to stop at a solution before not converging fully (the solver reached the maximum number of function iterations). This is why there are not all the data point for the query two graphs.

After gathering the different summaries, we ran the three queries using the α values to calculate the expected value of the count or the average. For the queries involving `COUNT(*)`, the expected value of a tuple with attributes $A = a_1$ and $B = b_1$ is

$$n * \alpha_{[A=a_1]} * \alpha_{[B=b_1]} * \alpha_{[A=a_1 \wedge B=b_1]},$$

where $\alpha_{[A=a_1 \wedge B=b_1]} = 1$ if that double attribute value is not computed.

For the query involving `AVG()` and a tuple with group by attribute $A = a_1$, the expected average of B is

$$\frac{\sum_{b \in B} b * \alpha_{[B=b]} * \alpha_{[A=a_1 \wedge B=b]}}{\sum_{b \in B} \alpha_{[B=b]} * \alpha_{[A=a_1 \wedge B=b]}},$$

where $\alpha_{[A=a_1 \wedge B=b_1]} = 1$ if that double attribute value is not computed.

Fig. 3a plots the EMD for the three queries versus the size, and Fig. 3b plots the EMD for the three queries versus the execution time. As a reference point, the EMD distance when assuming a uniform distribution for query 1 is 0.456, query 2 is 0.810, and query 3 is 0.113. The reason query 2 has a different uniform distance for this method versus the sampling method is because this method looks at all possible tuples, not just the ones in the truth set. So, the number of bins is significantly larger for query 2. For query 1 and query 3, the sampling methods produce all possible origins, which is why the uniform distance is the same.

As you can see, for query 1, there is only one data point because the minimum number of bins used is the number of bins needed for a single attribute histogram on each attribute. Since the query only involves a single attribute, the smallest result size is already the largest possible number of α's. For query 3, there are fewer data points because of the solver time problem explained above. Also not that the runtime and size axis are significantly smaller than those for the sampling methods.

For query 2, the EMD decreases as the size increases, as expected. The execution time seems to vary drastically; however, the axis is from 0 to 0.07 seconds, meaning the query executes instantaneously and we are likely seeing natural variation in execution times. For query 3, the EMD stays relatively constant for an increasing size and execution time. The reason for that is because of how we gather our original histogram selectivity values. Recall that query 3 involves an average of the actual elapsed flight time. When we gathered double attribute selectivity values, we were looking at those that contributed most to the total number of origin state and actual elapsed time pairs. We were not looking at those that contributed the most to the average elapsed time for each state. We hypothesis that if we changes how we were gathering our original selectivity values, our EMD would improve.

5. RELATED WORK

Although there has been work in the theoretical aspects of probabilistic databases [10], as far as we could find, there is not an existing, functional probabilistic database management system. This means our idea of using one to summarize an existing database is novel. However, there has been work by Markl [6] on using the maximum entropy principle to estimate the selectivity of predicates. This is similar to our usage of the maximum entropy principle except we are allowing for multiple predicates on an attribute and are using

the results to estimate the likelihood of a tuple being in the result of a query rather than the likelihood of a tuple just being in the database.

Our work is also similar to that by Suciu and Ré [8] except their goal was limited to estimating the size of the result of a query rather than tuple likelihood. Their method also relied on statistics on the number of distinct values of an attribute whereas our statistics are based on the selectivity of each value of an attribute.

There has been much research work in the past that uses sampling techniques for faster querying of large databases. In the work by Chaudhiri et al. [5], they precompute the samples of data that minimize the errors for the queries that can occur due to variance in the data. Here, they precompute the samples of data for a specific set of queries that they predicted. However, the later work of BlinkDB [4] improves on this by removing any assumptions on the actual queries. BlinkDB only assumes that there is a set of columns that are repeatedly queried, but the values for these columns can be anything among the possible set of values. BlinkDB then computes samples for each possible value of the predicate column in an online fashion when the user enters the query. We mainly refer to the implementation of BlinkDB for our work, though our work involves summarizing a database which is commonly a offline preprocessing step.

6. FUTURE WORK

For future work, we would like to test our methods on a more diverse range of queries, including ones involving other aggregate functions, selection predicates, and joins. Particularly with joins, we are aware that if you join two sampled relations, the result is often poor. The common solution is to either precompute the join and store samples or to sample one relation and join against the entirety of the other relation. However, this would bias the results towards those common in the sampled relation and would also result in slower runtime. We are curious to look into how stratified sampling and the probabilistic approach do when calculating joins.

Additionally, we would like to see what the results are without using any numerical return values like a count or average. One of the main motivations behind using the EMD was to be able to compare actual tuples, not just aggregate values. The benefit of sampling is that the entire tuple is already stored; so, retrieving it is not hard. With the probabilistic approach, we only compute and store the attributes necessary in the query. Also, we only store distinct values, meaning we have no way to reconstruct the discarded values.

One solution to this is to store a sample for each unique tuple in our predicate selection so that those tuples can be returned to the user. However, this drastically increasing the space required, and unless we made the samples extremely small, this would be no different, and likely worse, than stratified sampling. Having small samples, though, may still be a viable option.

Another major aspect to our future work is to decrease the runtime of the probabilistic method's solver. Having an execution time of 4.5 hours is completely unrealistic in most settings, especially if the EMD is not low enough to warrant such a long runtime. Possible solutions to this are to look for other, more efficient solvers or to implement our own. There may be some aspects of the particular problem we are solving that lend themselves to optimizing the solving process.

We would also like to investigate different methods for gathering the selectivity values used in the probabilistic approach. As explained in the evaluation section, just choosing those that contribute the most to the number of tuples of each attribute pair is not ideal when computing something other than a count. We need to look into a way of choosing a variety of selectivity values to improve the EMD on non-count queries. Potentially, a more dynamic approach is the best.

Finally for the sampling technique, currently we do a random sampling for each unique QCS value. Studies [5] have shown that random sampling might not work when the dataset has many outliers. So it would be worthwhile to investigate smarter methods of sampling rather than random sampling for each QCS value.

7. CONCLUSION

In this work, we proposed three different techniques to produce a summary of a large dataset. We evaluate the techniques on a flight dataset and compare the summarized relation size and query execution time to the Earth Mover's Distance for three potential data exploration queries.

In the end, for a highly repetitive dataset, such as the one we chose, the more advanced sampling technique does not seem to be beneficial to simple, non-selective queries. However, if the queries became more complex or the dataset less repetitive, the stratified sampling would have better results than the random sampling.

The probabilistic method proved to provide a large space and time savings, but you sacrificed more in terms of query correctness and preprocessing time. Both sampling techniques provided more correct query results and quicker preprocessing times, but required more space and time to query. This just goes to show that the best method of summarization is highly dependent on the user's requirements and the dataset. Just like with RDBMS's, a one size fits all approach may not be best.

8. REFERENCES

[1] `http://www.transtats.bts.gov/`.

[2] `http://docs.scipy.org/`.

[3] `https://pypi.python.org/pypi/pyemd/0.0.7`.

[4] Sameer Agarwal, Barzan Mozafari, Aurojit Panda, Henry Milner, Samuel Madden, and Ion Stoica. Blinkdb: queries with bounded errors and bounded response times on very large data. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 29–42. ACM, 2013.

[5] Surajit Chaudhuri, Gautam Das, and Vivek Narasayya. Optimized stratified sampling for approximate query processing. *ACM Transactions on Database Systems (TODS)*, 32(2):9, 2007.

[6] Volker Markl, Nimrod Megiddo, Marcel Kutsch, Tam Minh Tran, P Haas, and Utkarsh Srivastava. Consistently estimating the selectivity of conjuncts of predicates. In *Proceedings of the 31st international conference on Very large data bases*, pages 373–384. VLDB Endowment, 2005.

[7] Frank Olken. *Random sampling from databases*. PhD thesis, University of California, 1993.

[8] Christopher Ré and Dan Suciu. Understanding cardinality estimation using entropy maximization. *ACM Transactions on Database Systems (TODS)*, 37(1):6, 2012.

[9] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. The earth mover's distance as a metric for image retrieval. *International journal of computer vision*, 40(2):99–121, 2000.

[10] Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. Probabilistic databases. *Synthesis Lectures on Data Management*, 3(2):1–180, 2011.