A Large-Scale Real World Data Integration Use Case Analysis for DuBio

DAVID HESTHAVEN, University of Twente, The Netherlands

This research focuses on the limitations and execution performance of semantic duplicate cleaning on real-world data of the probabilistic database DuBio, which was developed at the University of Twente. This was done by using the WDC Product Data Corpus, a large collection of data, to find the overhead of running similar queries on two versions of the same database, one probabilistic and one not. The goal is to find how increasing the size of the database or the size of clusters within the database, affect the relative difference in overhead between the two versions of the database, alongside finding any additional aspects that may influence the overhead.

Additional Key Words and Phrases: DuBio, Probabilistic Databases, WDC Data Corpus, ID-Cluster.

1 INTRODUCTION

In recent years, there is a consistent increase in the desire for tools to aid in data cleaning [2], as the large amount of data gained by scaping the internet or other methods often results in large quantities of data with high uncertainties.

For this reason, the DuBio extension was developed at the University of Twente, to help generate probabilistic databases which aid in improving data cleaning and data quality. However, while the DuBio extension is functional, its limitations have yet to be tested thoroughly, and the goal of this research is to find those limits. This will be done by utilizing the WDC Data Corpus, a large scale collection of data from across the internet, which grouped items it believed to be the same into specific ID-clusters. These clusters will be organized by their sizes and placed into database tables containing a certain number of clusters, randomly selected from the main set. By utilizing this data corpus and the methodology outlined below, this research intends to find the overhead of using DuBio, along with any bottlenecks that may exist within the extension.

2 BACKGROUND

2.1 Probabilistic Databases

Due to the nature of data, many databases possess uncertainty within the data [1], and probabilistic databases exist to aid in solving this issue. Within probabilistic databases, entries are kept in various different states. As an example, if a particular attribute had a probability of either 1 or 0, it could be viewed as that entry having two states, one where it exists in the database, and one where it does not. Rather than selecting a single state, probabilistic databases return a list of the various possibilities alongside their probabilities. Probabilistic databases are an active field of research to which this project hopes to contribute. DuBio is an extension that seeks to aid in Probabilistic Data Integration (PDI) particularly through the stage of continuous improvement within the data integration process [6], and this research will test how effective it is in accomplishing that task. In DuBio, information on entries is stored within a "sentence" such as "x=2 and y=1". To explain these "sentences" in more detail, consider a situation where there exist two entries, then there exists two possibilities: either the entries are different, or they are the same. These two possibilities can be represented by a variable, with "x=1" representing the two entries being different, and "x=2" representing the two entries being the same. If the two entries are the same, then if they have two different prices, for example, then one must consider the two possibilities of which of those two prices is the correct one, which can be represented by another variable, like "y". So the sentence "x=2 and y=1" implies that the two entries are in fact the same, and that the price of the first entry is the correct one. All of these variables can then be referenced in a dictionary to determine what the values of those variables aught to be, and by extension, what the probability of that entry appearing in the results should be.

2.2 WDC Product Data Corpus

The WDC Product Data Corpus or as it will henceforth be called, the WDC, is a truly massive dataset of over 26 million different product offers from approximately 79 thousand websites. The WDC was developed by A. Primpeli et al. [5] Utilizing their algorithm, these products would be grouped together into an ID-Cluster if the algorithm believed any group of products were in-fact the same product. In total, there exist approximately 16 million ID-clusters. Of these, only 4,400 have been manually verified as either matching or not matching. The size of an ID-cluster can vary wildly, with the smallest being of size 2, and the largest being above 80. For this research, we will be utilizing the English subset of the WDC, which only includes products from English related top-level domains, such as .com, .net, .co.uk, .us, and .org. This reduces the size of the set to approximately 16 million offers from 43 thousand websites sorted into approximately 10 million clusters. Additionally, for simplicity's sake, the normalized dataset will be utilized, in which all values are lowercased and all non alphanumeric characters have been removed.

3 RESEARCH QUESTION

The main research question this research hopes to resolves is this: *What is the performance and limitations of the DuBio extension with regards to scalability and execution performance?* This is then split into two sub-questions to aid in answering the main question.

- RQ1: What is the rate at which the runtime of DuBio increases as the size of a given ID-cluster increases?
- RQ2: How does DuBio scale in runtime as the number of ID-clusters of the same size are given to it?
- RQ3: How can probabilistic data be generated in DuBio for a cluster of records that represents all possible merge outcomes when the record matches in the cluster are assumed uncertain?

TScIT 41, July 5, 2024, Enschede, The Netherlands

4 ALGORITHMIC DEVELOPMENT

In this section we will review the development and usage of the algorithms developed for this research, specifically with the goal of answering RQ3. All code referenced in this paper can be found at the github link in the references [4], alongside a README to explain it's setup and usage in detail. The code primarily used the Psycopg2 library for Python as it is the easiest way to interface with the PostgreSQL database. First, a basic piece of code was written to convert the large .json file which contained all the data of the WDC into a database table. This was primarily based off of old code, and as a result, took the smallest amount of time relative to the rest of the code development. The primary changes necessary were due to the fact that the old code had been written to handle an older version of the WDC data. The new data simplified things significantly, as in the old version, each product did not have its own unique ID, so one had to be created using various attributes as the primary key, coupled with a few other unique decisions by A. Primpeli et al. [5] which made the original data more complex. The new version was simplified and had each product having its own unique ID, allowing for that to be used as the primary key.

The primary difficulty in development was handling the myriad possibilities when attempting to merge a cluster ID. This can be seen clearly by looking at the increasing complexity as cluster size increases. At cluster size 2, there are only two possibilities, either the records are the same, or they are different. At cluster size 3, there are now five possibilities, increasing further to 15 different possibilities at cluster size 4. A complete list of the possibilities for each cluster size and how records are grouped can be found in appendix C. This becomes even more complex when the possibilities of price being NULL is introduced. A price being NULL is considered as being an unknown, so in the situations where two records are considered the same, if one of the prices is NULL, then it is assumed that the not NULL price is the correct one. The algorithm works by getting all unique cluster IDs, and then getting the records for each cluster ID. These records are then compared to see which prices are NULL, which are not, and whether the prices that are not NULL are the same or different. The correct possibility is then selected via a long series of if and elif statements and then the correct results would be output as they were calculated out in advance and then hard coded into each statement. As a result, each block of code is very similar, effectively the same lines copy and pasted with small numbers changing depending on the situation. Because of this, the technical difficulty of developing these algorithms was not particularly high, but it did require a high level of meticulousness and attention to detail. Since everything in the algorithms is hard coded, the methods only require the names of the base table from which to get the records, the probabilistic table that it will fill out, and the name of the dictionary that the user would like the variables to be stored in.

To give an example of a complex cluster and the sentences that would be generated from it, consider a cluster of size 4 with the four records A, B, C, and D. Then assume a situation where the prices of A and C are the same, B is NULL, and D is not NULL, but different from A or C. The system defaults to allow the earliest record be the dominant one if possible, as a result, in all of the possibilities where A and C were assumed to be the same, it would be stated that A would exist instead of C. Thus, the record A will exist in all possibilities where AC were considered the same, and all possibilities where AB were considered the same due to B being NULL, and the one situation where ABC were considered the same, leading to a sentence of length 10. A would have a sentence that clarifies it exists in the 10 possibilities of: 1, 2, 5, 6, 7, 8, 9, 10, 11, and 14. Similarly, since B is NULL, records C and D will also gain an additional two possibilities where they always exist, and B will only exist in the base five possibilities where B is considered wholly separate from the others. This then leads to the more complicated aspect, that is, what happens when two different prices have to be merged, like AD or CD. In the situations where only two different prices are merged, an additional two records are generated for that situation acting as the x value in the sentence. A y variable is then generated and added to the sentence, one for each price possibility. They are assumed to have an equal chance of being the correct price, so each y value has a probability of 0.5. Then we come to the final problem of the situations like ACD or ABCD. These are handled the same way, as in the latter situation, B is disregarded due to the price being NULL. Like before, two new records are generated for the new possibilities, but the variable being generated is z instead of y to keep them separate from each other. Since there are two instances of price A, and only one of price D, the probabilities are instead set to 0.67 and 0.33 respectively for the z values. As can be seen, the various possibilities have become very complex an convoluted already by the time cluster size 4 is reached.

The use of hard coding these possibilities made the implementation of the code itself quite simple, but had the side of effect of as the size of the cluster got larger, the more complex the code became, resulting in more time needing to be dedicated to making sure everything is perfect. Therefore, using this method to work with larger cluster sizes is not viable in our opinion. We only reached up to cluster size 4, and while sizes 5 and 6 might still be able to be done, it would be extremely strenuous and is not recommended. If further research is to be done with regards to higher cluster sizes, a new coding method must be used.

5 METHODOLOGY

5.1 Setup

In order to properly perform the queries on the various database tables to gather data, the database must first be prepared appropriately. First, the main database table must be added to the table using a piece of code that reads the base dataset from the json file provided by A. Primpeli et al. [5] and inputs it into a main table. This is a slow process, taking on average 5-6 hours. Next, the database is organized by utilizing a series of prompts, the complete versions of which can be found in Appendix A. First a table including all cluster IDs and their sizes needs to be created and filled out (A.1).

Then a list of all cluster IDs of a specific size needs to be created and ordered in a random fashion (A.2). Finally, the base table needs to be filled out with a certain number of clusters of a given size (A.3). Then, using the appropriate code, a probabilistic table will be filled in, alongside a dictionary being filled out in the appropriate dictionary table. An SQL statement to generate a template for the base and probabilistic tables can be found in Appendix A.4 and A.5 respectively.

In the WDC, each product has a large number of attributes, which complicates things with regards to converting a table into a probabilistic one. As a result, the only attributes that are maintained to the probabilistic tables are cluster ID, category, and price. Cluster ID is there for self-explanatory reasons, while category and price were deemed to be interesting attributes worth examining for this research. Category was a universal attribute, in that it did not have any NULL values, though it did have a "not found" value which fulfilled the same situation. Additionally, the category attribute was uniform, with all products of a cluster always possessing the exact same category value. Price, meanwhile, is significantly more random, and therefore acts as the primary driver for whether or not a product has multiple different possible entries.

5.2 Gathering Data

Now that both versions of a table with a specific cluster size and number exist, they can be queried and the overhead of the two can be determined. Examples of the four different queries used can be found in Appendix B, with the table being queried being one where the cluster size is 2, and the number of clusters is 100. You may notice in the appendix that the queries appear different, but that's not the case. They are the same queries, it's simply that the implementation differs between the base and probabilistic. The first query (B.1) seeks entries where the price attribute contains the string 'usd'. The second query (B.2) seeks entries where the price attribute is not NULL. The third query (B.3) seeks entries where the category is 'not found' which is effectively the version of NULL for the category attribute. The fourth query (B.4) seeks entries where the length of the string in the category attribute is longer than 10 characters long. The reason for this, is that this selects all entries where the category contains more than one word in the category attribute. The base table is then queried 10 times for each query, the time taken is recorded and then averaged for each query. The same is then done for the probabilistic table. Then the relative difference between the two times is calculated and recorded. There was a change in methodology partway through the research due to the discovery of a limitation in DuBio.

Initially, the cluster counts that would be tested were 100, 250, 500, 1000, 2500, 5000, 6000, 7000, 8000, 9000, and 10000. However, due to an unexpected limitation of DuBio, which will be discussed further on, difficulties were discovered at cluster size 4, after cluster sizes 2 and 3 had been completed. As a result, the cluster counts used were changed to 100, 250, 500, 750, 1000, 1500, 2000, 2500, 3000, 3500, and 4000, as 4000 was the highest round cluster count that was usable for size four clusters. The experiment was then redone for sizes two and three for the counts that we didn't have data for. All data gathered, including those from the initial methodology for sizes two and three can be found int he google sheets page in the references [3].

5.3 Metrics

In this research, two primary metrics were measured. The first was "Relative Difference" which represented the relative difference in time between the average performance time of the base database query, and the average performance time of the probabilistic database query. The formula used to calculate this metric is:

$$y = t_2/t_1 \tag{1}$$

In which t_1 represents the time of the base query, and t_2 represents the time of the probabilistic query. This metric is used in Figures 1, 2 and 3 in the results section.

The second metric that was measured was the "Rate of Change" metric, which represented the rate of increase in performance time between two different cluster sizes. The formula used to calculate this metric is:

$$z = y_{(n+1)}/y_n \tag{2}$$

In which y_n represents the smaller cluster size, and y_{n+1} represents the larger cluster size, as we are only comparing differences in cluster size of 1. This metric is used in Figures 4 and 5 in the results section.

6 RESULTS

The first set of results that we will look at is the relative difference in overhead on tables with cluster size 2, which is shown in Figure 1. To explain what exactly is meant by relative difference and the y value in all following graphs, if y=50, then that would mean it took the probabilistic table 50 times as long on average to process its query than it took the base table to process.

From Figure 1, two things can immediately be seen. First, as the number of clusters in a table increases, the relative difference in time also increases, at what appears to be a linear rate, though there are aberrations in the data, most likely due to the sample sizes of 10 not being enough to create a perfect curve, but enough to verify the trend, which shows a strong linear correlation between cluster count and relative overhead. The second thing that can be clearly seen is that the query itself has a drastic impact on the relative difference, with Query 3 having a much lower relative difference than the other queries, particularly 2 and 4. The reason for this is not currently known, though a theory will be examined in the discussion section.

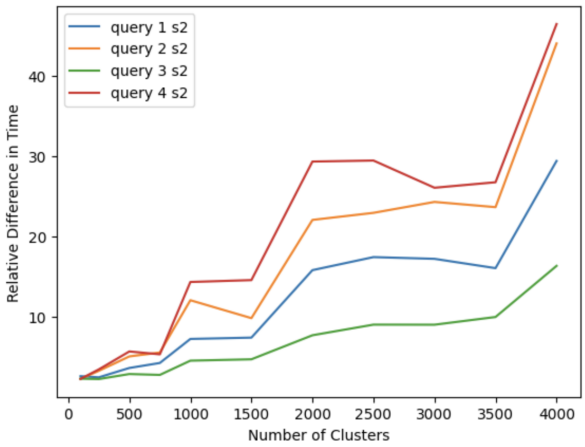

Fig. 1. Relative Difference of Cluster Size 2.

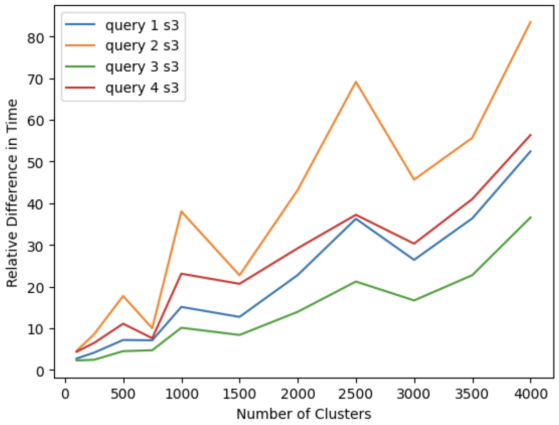

Fig. 2. Relative Difference of Cluster Size 3.

Now to see if the this holds for other sizes, observe Figure 2. which shows the trends for tables made up of clusters of size 3.

Again, the two initial conclusions from Figure 1. hold, in that as the number of clusters increases, so does the relative difference in time, at a seemingly linear rate, and different queries have a different impact on that relative difference, however a few new things of note makes themselves known. First, the trend appears even more erratic than at size 2, particularly for query 2, but overall, the trend still remains. Second, which queries have the highest relative difference does not remain consistent, with Query 4, which previously was very similar to Query 2 in its relative difference, is now significantly less than Query 2, and is now very similar to Query 1 instead. It should be noted that it may appear that this data starts at a cluster number of 0. It does not, it starts at a cluster number of 100, it's simply that due to the large final scale of the x-axis, it appears to start at 0 even though it does not.

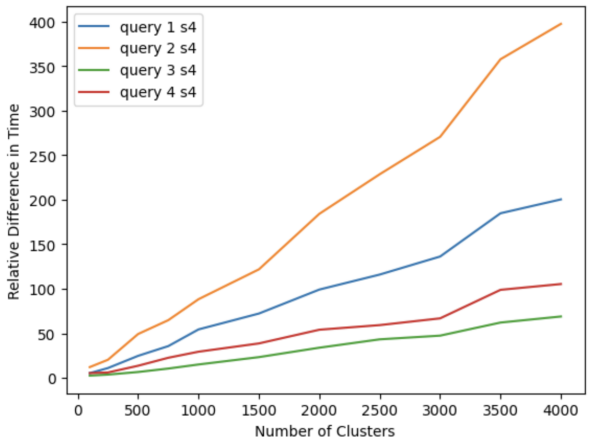

Fig. 3. Relative Difference of Cluster Size 4.

Next, let us look at Figure 3. which is the same as the previous two graphs but for cluster size 4.

This graph shows a very clear and clean linear correlation, the cleanest of all the graphs thus far. This may imply that the erratic behaviour in the previous two graphs is due to the fact that half of the data in those graphs was taken at different points in time due to the change in methodology, however due to time constraints, it was not feasible to go back and redo the entire experiment, though this should be done in future to verify results. Second, we see the trend of which queries increase in relative difference, with Query 2 now becoming extremely dominant, and Query 4 dropping below Query 1 and now being much closer to Query 3, even though previously they were on opposite sides of the graph. The exact reasoning for why the queries act the way they do is beyond the scope of this research, however, a theory does exist for why, which will be discussed in the discussion section.

Now, let us look at Figure 4. which graphs the rate of change between cluster sizes 2 and 3.

This graph shows two trends of significance. First, that at low numbers of clusters, the relative difference between the two sizes is significantly more erratic, shown primarily by queries 2 and 4, but as the number of clusters increases, the relative difference appears to stabilize to some degree. Second, the query likewise has an impact on how significant the increase in cluster size has on the relative difference in overhead, with Query 4 seeming to increase the relative difference by a factor of approximately 1.4, while Query 3 increase the relative difference by a factor of approximately 2.5, a rather significant gap.

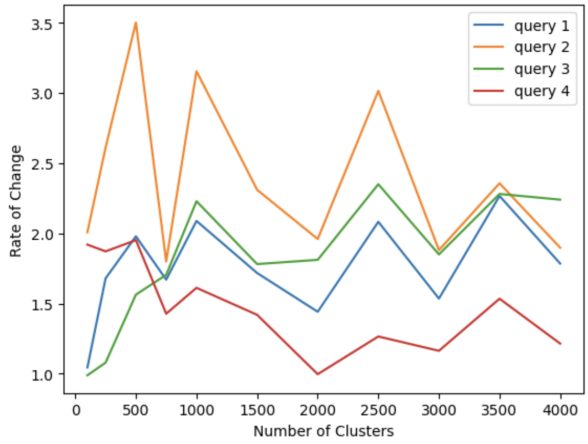

Fig. 4. Relative Difference Between Cluster Sizes 2 and 3.

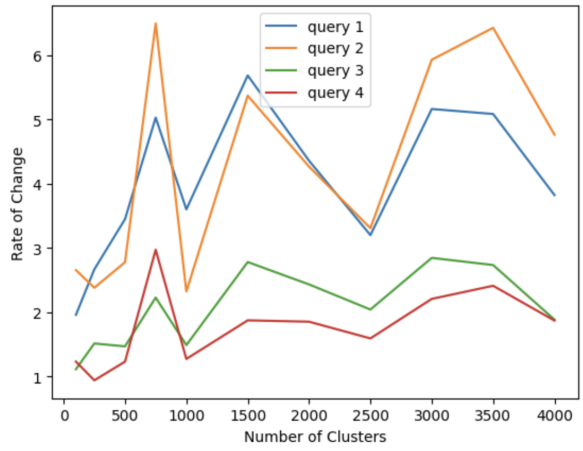

Fig. 5. Relative Difference Between Cluster Sizes 3 and 4.

Finally, let us now observe Figure 5. which is the same as Figure 4. but for cluster sizes 3 and 4.

This graph displays similar results to the previous, with a few new things of note. Similar to the previous, the results are significantly more erratic at lower cluster counts, but seem to stabilize somewhat at higher values. However, where they stabilize has changed. Queries 3 and 4 appear to stabilize at a y value of around 2, implying a doubling of relative difference, but Queries 1 and two seem to stabilize somewhere between 4 and 5, a much higher relative difference. Something worth noting is that Queries 1 and 2 are focused on the attribute of price while Queries 3 and 4 are focused on the attribute of category. This fact is a significant part of the theory as to why the queries act the way that they do which will be elaborated upon in the discussion section.

With this data, we can now attempt to answer the research questions posed at the start of this research. First, as the size of a cluster IDs increases, the overhead will increase to some degree, however it is not consistent as to how much the overhead will increase by. Additionally, the queries have a significant impact on how much the overhead increases by. By comparing Figures 4 and 5, we can conclude that Queries 3 and 4 seem to increase at about a factor of two each time, implying that for these two queries, increasing cluster size by a value of n will likely result in a relative increase of 2^n. However, there is a significant change for Queries 1 and 2, and therefore a trend cannot be determined at this time, beyond that as cluster size increases, overhead does as well.

Second, as the number of clusters in a table increases, the overhead will increase at a linear rate, though what the scalar multiple of that linear rate is exactly is not consistent. Third, it has been discovered that the queries used on the tables has as much an impact on overhead as cluster size or number if not more so, with cluster size having a different increase depending on the query, and the scalar value for the linear increase of overhead in the case of cluster number appears to be closely tied to the query, which leads to the next section.

Noted here is the reason that the methodology with regards to cluster count had to change partway through the research. When attempting cluster size 4, cluster count 5000, an error occurred within the dictionary table. The reason for this, is that within DuBio, there is a dictionary that is stored within a "_dict" table, from which the various probabilities need to be drawn in order to calculate probabilities. Unbeknownst to us at the start of the research, this dictionary can only hold 16 bits worth of variables, or 65,536 variables within it. Due to the increasing complexity of the clusters, this limit was reached at cluster size 4, meaning the full methodology could not be carried out and had to be changed in order to properly compare the different cluster sizes.

7 DISCUSSION AND FUTURE WORK

The data gathered over the course of this research showed a trend that was not initially considered at the start of this research, namely the significant impact that a query has on the overhead of the usage of DuBio. Why exactly this occurs is not known precisely, as it is outside the scope of this research, however, due to data collected in the course of the research, a hypothesis has presented itself which can be a starting point for future research.

The initial hypothesis was simply that the amount of results a query returns is directly related to the relative difference, however this turned out to be incorrect, though it did have significance. At lower cluster sizes (2 and 3), Query 4 always had the longest runtime for the probabilistic table, but not necessarily the highest relative difference, even though it did the highest number of results. Cluster size 4 is what helped develop this hypothesis, as it showed Query 2

completely overtaking Query 4, not just in absolute time but also in how many results it would return. Thus, the current hypothesis is that the influence a query has on the relative difference is directly related to the relative difference between the number of results in the base query versus the number of results in the probabilistic query. To use an example, at cluster size 4, count 4000, Query 2 would return 2815 results for the base query, but 15472 for the probabilistic one, a factor of approximately 5.5. Compare this to Query 4, with an initial result of 9516 and a final result of 11747, merely a factor of approximately 1.25. Since Query 2 has such a massive increase in the number of results, it obtains a much greater relative difference than Query 4 does.

As to why this occurs, it is most likely due to the fact that Query 4 is focused on the "category" attribute while Query 2 is focused on the "price" attribute. Since category is unchanging within a cluster, the initial results are far more frequent than in Query 2, as price is NULL approximately 80% of the time. But when price is not NULL, there is a much higher chance that there will be additional entries, and since Query 2 is specifically looking for prices that are not NULL, it finds far more entries in the probabilistic table, and this ratio only increases as cluster size increases, as if there are 4 different prices, then that means there will be 44 entries in the probabilistic table as opposed to merely 4 in the initial. This is also seen in how Query 1, which also focuses on price, similarly has a much higher relative difference as cluster size increases, while Query 3, which also focuses on category, has a very low relative difference increase, as it also has the lowest relative difference between initial and final of all queries. If this hypothesis is true, then Query 3 actually has a great deal of value, as it is the query that is closest to showing the "raw" overhead of calculating probability. We do not have enough data to definitely prove this hypothesis, but are confident enough in our current data that even if it is not wholly correct, it is likely to be at least a partial answer.

Now it is worth looking at what directions future research can go in. The first is clear, extending the current research with larger cluster counts and sizes. With regards to counts, we don't believe that it is a particularly useful field, in that it will only really be useful to confirm our current results. In that case, we would recommend focusing on repeat trials as much as possible to attempt to minimize the erratic data and trends we see in our data. As for cluster sizes, this is a more interesting field in our opinion, though as we stated at the end of section 4, the current method is not viable for larger cluster sizes due to the ever increasing complexity, so a new method would have to be developed to truly see the trends of increasing cluster size. Additionally, more cluster sizes would also allow for other kinds of research, such as selecting a set of 100 clusters, each of which can be randomly size 2 or 3, and seeing how that influences things. What happens when the clusters can be sizes 2, 3 or 4? This is an interesting field of research, in our opinion, though it would require the ability to convert larger cluster sizes as well. The current implementation could be extended to be used for cluster size 5 as well, and at that stage such research could produce good preliminary results. However, what we believe is the most interesting field of research to continue in, would be trying to find the significance that a query has on the overhead in more express, quantifiable terms, in the attempt to prove or disprove the hypothesis above.

Finally, we would like to make two recommendations to the developers of DuBio, should they wish for further research like this one. First, alter the limitations on the variable dictionary so that it is no longer limited to a 16 bit integer. Otherwise, larger sizes and cluster counts will simply not be able to be tested, and DuBio will remain an extension that can only be used for smaller scale probabilistic databases. There isn't anything wrong with that, but in that case, there is no real need for further research of this variety. Second, when the question of why the queries acted the way that they did first came up, we attempted to understand by using the EXPLAIN ANALYZE command within PostgreSQL. This did have a use for our research, as it showed a nested loop, which only ran once, but took up the overwhelming majority of the time of the query. From this, we were able to conclude that this nested loop is where the probability calculations are handled, and that these probability calculations are indeed the most significant aspect of the query compared to the sorting or filtering, proving the validity of our data. An example of this plan can be found in Appendix D. However, what is actually going on within that nested loop remains a black box. We have no idea what part of the calculations is taking the most time and such. We therefore recommend the developers to see if a method to allow PostgreSQL to see and measure the time it takes various parts of the extension to run, so that further data can be collected with the goal of improving DuBio.

8 CONCLUSION

The data collected over the course of the research has yet to perfectly answer all research questions set out at the beginning of this research, but it has provided significant evidence to display the limitations of the DuBio extension alongside showing viable new avenues for research. We have shown that the relative overhead increases at a linear rate relative to the number of clusters in a table, and that overhead increases at a seemingly steady rate as cluster size increases, though that rate appears dependant on the query being posed to the database. Finally, we have shown that queries have a significant impact on overhead, and are of the opinion that further research into the queries should be the next stage of finding the limitations of the DuBio extension.

ACKNOWLEDGMENTS

I woudl like to thank my supervisor, Maurice van Keulen, for his assistance throughout the research and his time and effort to give feedback both in meetings and at various points throughout the research period. Additionally, I would like to thank Matteo Schut and Aren Merzoian for giving feedback on this paper.

9 AI USAGE

AI was not used at any point during this research, not to develop code, nor collate data, nor write any part of this report.

REFERENCES

[1] Charu C. Aggarwal and Philip S. Yu. 2009. A Survey of Uncertain Data Algorithms and Applications. *IEEE Transactions on Knowledge and Data Engineering* 21, 5 (2009), 609–623. https://doi.org/10.1109/TKDE.2008.190

[2] Nilesh Dalvi, Christopher Ré, and Dan Suciu. 2009. Probabilistic Databases: Diamonds in the Dirt. *Commun. ACM* 52 (07 2009), 86–94. https://doi.org/10.1145/1538788.1538810

[3] David Hesthaven. 2024. *DuBio Scalability Testing Raw Data Collection.* https://docs.google.com/spreadsheets/d/1tJg4qRd8DJ3VyEi49l5S8Rk7lQt2wVIhm8wb1RJhtig/edit?usp=sharing

[4] David Hesthaven. 2024. DuBio-WDC-Scalability-Testing. https://github.com/DavidBH98/DuBio-WDC-Scalability-Testing.

[5] Anna Primpeli, Ralph Peeters, and Christian Bizer. 2019. The WDC Training Dataset and Gold Standard for Large-Scale Product Matching, In The WDC Training Dataset and Gold Standard for Large-Scale Product Matching. *WWW '19: Companion Proceedings of The 2019 World Wide Web Conference*, 381–386. https://doi.org/10.1145/3308560.3316609

[6] Maurice van Keulen. 2018. *Probabilistic Data Integration.* Springer, Germany, 1–9. https://doi.org/10.1007/978-3-319-63962-8_18-1

A SETUP SQL QUERIES

A.1 Appendix A.1

INSERT INTO wdc_eng_cluster_size SELECT cluster_id, COUNT(*) AS count FROM wdc_eng_offers GROUP BY cluster_id;

This statement inserts all unique cluster IDs alongside its sizes into a table "wdc_eng_cluster_size".

A.2 Appendix A.2

INSERT INTO wdc_eng_clusters_x (cluster_id) SELECT cluster_id FROM wdc_eng_cluster_size WHERE number=x ORDER BY RANDOM();

This statement selects all clusters of a specific size, and randomly sorts them into a new table "wdc_eng_clusters_x" where x is the cluster size.

A.3 Appendix A.3

CREATE TEMP TABLE wdc_eng_n2_temp(cluster_id BIGINT NOT NULL, PRIMARY KEY(cluster_id));
INSERT INTO wdc_eng_n2_temp SELECT * FROM wdc_eng_clusters_2 LIMIT 100;
INSERT INTO wdc_eng_ex_s2n100_base SELECT wdc_eng_offers.category, wdc_eng_offers.cluster_id, wdc_eng_offers.id, wdc_eng_offers.price FROM wdc_eng_offers, wdc_eng_n2_temp WHERE wdc_eng_offers.cluster_id=wdc_eng_n2_temp.cluster_id;

This set of statements first creates a temporary table, and fills it with a limited number of cluster IDs of a specific size, in this case 100 cluster IDs of size 2. This is then used to fill the base table with the entries of products belonging to those 100 cluster IDs.

A.4 Appendix A.4

CREATE TABLE base_template (category TEXT, cluster_id BIGINT NOT NULL, id TEXT NOT NULL, price TEXT, PRIMARY KEY(id));

This statement is a basic setup statement to create a base table including the category, cluster_id, id, and price attributes.

A.5 Appendix A.5

CREATE TABLE prob_template (index BIGINT NOT NULL, nid BIGNIT NOT NULL, cluster_id BIGINT NOT NULL, category TEXT, price TEXT, _sentence BDD, PRIMARY KEY(index));

This statement is a basic setup statement to create a probabilistic table including the category, cluster_id, and price attributes from the previous tables, and an index, nid, and _sentence attribute as the new ones needed for the probabilistic table.

B DATA GATHERING SQL QUERIES

B.1 Query 1

This query selects all entries where the price contains "usd" in it.

B.1.1 Base Query. SELECT t1.id, t1.cluster_id, t1.category, t1.price FROM wdc_eng_ex_s2n100_base t1 WHERE t1.price LIKE '%usd%' ORDER BY t1.cluster_id;

B.1.2 Probabilistic Query. SELECT t1.nid, t1.cluster_id, t1.category, t1.price, t1._sentence, prob(d.dict, t1._sentence) AS probability FROM wdc_eng_ex_s2n100_prob t1, _dict d WHERE d.name = 's2n100' AND t1.price LIKE '%usd%' ORDER BY t1.index;

B.2 Query 2

This query selects all entries where the price is not NULL.

B.2.1 Base Query. SELECT t1.id, t1.cluster_id, t1.category, t1.price FROM wdc_eng_ex_s2n100_base t1 WHERE t1.price IS NOT NULL ORDER BY t1.cluster_id;

B.2.2 Probabilistic Query. SELECT t1.nid, t1.cluster_id, t1.category, t1.price, t1._sentence, prob(d.dict, t1._sentence) AS probability FROM wdc_eng_ex_s2n100_prob t1, _dict d WHERE d.name = 's2n100' AND t1.price IS NOT NULL ORDER BY t1.index;

B.3 Query 3

This query selects all entries where the category is "not found".

B.3.1 Base Query. SELECT t1.id, t1.cluster_id, t1.category, t1.price FROM wdc_eng_ex_s2n100_base t1 WHERE t1.category LIKE 'not found' ORDER BY t1.cluster_id;

B.3.2 Probabilistic Query. SELECT t1.nid, t1.cluster_id, t1.category, t1.price, t1._sentence, prob(d.dict, t1._sentence) AS probability FROM wdc_eng_ex_s2n100_prob t1, _dict d WHERE d.name = 's2n100' AND t1.category LIKE 'not found' ORDER BY t1.index;

B.4 Query 4

This query selects all entries where the length of the string in the category attribute is longer than 10 characters.

B.4.1 Base Query. SELECT t1.id, t1.cluster_id, t1.category, t1.price FROM wdc_eng_ex_s2n100_base t1 WHERE length(t1.category)>10 ORDER BY t1.cluster_id;

B.4.2 Probabilistic Query. SELECT t1.nid, t1.cluster_id, t1.category, t1.price, t1._sentence, prob(d.dict, t1._sentence) AS probability FROM wdc_eng_ex_s2n100_prob t1, _dict d WHERE d.name = 's2n100' AND length(t1.category)>10 ORDER BY t1.index;

C RECORD GROUPING POSSIBILITIES WHEN MERGING

In this appendix is shown how records were grouped in the various possibilities depending on cluster size. If two letters are separated by a gap, then they are considered as different records in that circumstance, but if they are directly next to each other, then they are considered the same record.

C.1 Cluster Size 2

- 1: A B
- 2: AB

C.2 Cluster Size 3

- 1: A B C
- 2: A BC
- 3: AC B
- 4: AB C
- 5: ABC

C.3 Cluster Size 4

- 1: A B C D
- 2: A BCD
- 3: ACD B
- 4: ABD C
- 5: ABC D
- 6: AB C D
- 7: AB CD
- 8: A B CD
- 9: AC B D
- 10: AC BD
- 11: A C BD
- 12: AD B C
- 13: AD BC
- 14: A D BC
- 15: ABCD

D EXPLAIN ANALYZE QUERY RESULT

QUERY PLAN
Sort (cost=2223.27..2247.15 rows=9552 width=248) (actual time=14550.231..14553.348 rows=11747 loops=1)
Sort Key: t1.index
Sort Method: external merge Disk: 3488kB
-> Nested Loop (cost=0.00..1591.81 rows=9552 width=248) (actual time=2.083..14516.723 rows=11747 loops=1)
-> Seq Scan on _dict d (cost=0.00..7.55 rows=1 width=774) (actual time=0.048..0.063 rows=1 loops=1)
Filter: ((name)::text = 's4n4000'::text)
Rows Removed by Filter: 48
-> Seq Scan on wdc_eng_ex_s4n4000_prob t1 (cost=0.00..1464.86 rows=9552 width=240) (actual time=0.015..31.239 rows=11747 loops=1)
Filter: (length(category) > 10)
Rows Removed by Filter: 16910
Planning Time: 0.288 ms
Execution Time: 14556.400 ms

The ".." within the plan represents start versus end time. So in the nested loop it says "2.083..14516.723" meaing it started the nested loop at 2 milliseconds, and ended it at 14.5 seconds.