informatics

MDPI

Article

Selective Wander Join: Fast Progressive Visualizations for Data Joins

Marianne Procopio [1,*], Carlos Scheidegger [2], Eugene Wu [3] and Remco Chang [1]

[1] Department of Computer Science, Tufts University, Medford, MA 02155, USA; remco@cs.tufts.edu
[2] Department of Computer Science, University of Arizona, Tucson, AZ 85721, USA; cscheid@email.arizona.edu
[3] Department of Computer Science, Columbia University, New York, NY 10027, USA; ewu@cs.columbia.edu
* Correspondence: marianne.procopio@tufts.edu

Received: 29 January 2019; Accepted: 14 March 2019; Published: 25 March 2019

Abstract: Progressive visualization offers a great deal of promise for big data visualization; however, current progressive visualization systems do not allow for continuous interaction. What if users want to see more confident results on a subset of the visualization? This can happen when users are in exploratory analysis mode but want to ask some directed questions of the data as well. In a progressive visualization system, the online aggregation algorithm determines the database sampling rate and resulting convergence rate, not the user. In this paper, we extend a recent method in online aggregation, called Wander Join, that is optimized for queries that join tables, one of the most computationally expensive operations. This extension leverages importance sampling to enable user-driven sampling when data joins are in the query. We applied user interaction techniques that allow the user to view and adjust the convergence rate, providing more transparency and control over the online aggregation process. By leveraging importance sampling, our extension of Wander Join also allows for stratified sampling of groups when there is data distribution skew. We also improve the convergence rate of filtering queries, but with additional overhead costs not needed in the original Wander Join algorithm.

Keywords: progressive visualization; online aggregation; interaction; information visualization

1. Introduction

Responsive visual exploration typically requires that the underlying data be available at interactive speeds. This requirement can be easily met when data resides in memory, but such a solution is not scalable. As the amount of data increases, the data must be stored on disk or in a remote database. As a result, the queries used to populate the visualization can take minutes, hours or longer to return, resulting in long wait times between each of the user's interactions and diminishing the user's ability to quickly explore the data.

This lag is exacerbated if the analysis queries involve joining data across multiple database tables. For example, consider the following database query that finds the average customer spending per region:

```
SELECT AVG(orders.order_total), location.region
FROM   orders, customers, location
WHERE  orders.customerID = customer.customerID AND
customer.locationID = location.locationID
GROUP BY location.region
```

The query must read data from three separate tables, combine them together using the expressions in the `WHERE` clause, partition the resulting data by `location.region`, and finally compute the average

`order_total` for each region. `JOIN` operations are well known to be costly [1]. To compute fully accurate results, the join operation requires at minimum full scans of all input tables, and naive implementations take quadratic time in the sizes of the tables.

Progressive visualization is a recent technique to adapt to the ever-increasing data processing latency problem. At its core, Selective Wander Join is a progressive visualization system that enables the user to prioritize data analysis of interest while performing exploratory analysis, optimized for queries involving aggregation across multiple tables in data joins. Individual components of the visualization are selected and the underlying data is weighted appropriately so that these records are sampled more often in the online aggregation process. This lets the aggregation values converge faster, but still allows for all components of the visualization to be sampled and estimated values updated.

To achieve this, we integrate the idea of importance sampling [2] into a recent online aggregation method, called Wander Join [3] and describe how to perform this prioritization. Additionally, by applying importance sampling to Wander Join, we can ensure even sampling of different groups when there is an uneven distribution of data across the grouping attribute. We can also show that changing weights of individual records can speed up convergence for queries involving filters. Importance sampling adds additional overhead and bookkeeping not found in the original Wander Join algorithm but we show its impact does not significantly affect performance and performs no worse than Wander Join.

Since Selective Wander Join leverages importance sampling to enable userdriven interactivity in the sampling process, we applied interaction techniques that allows the user to monitor and control the sampling rate of each group. This gives users more transparency into the online aggregation process and prioritizing their data of interest. We describe a visualization design leveraging common progressive visualization techniques to show how to apply the Selective Wander Join algorithm in Section 6 and present a demonstration study of the visualization with expert users in Section 7.

Through this work, we claim the following 3 contributions:

- Extended online aggregation methods to support joins for common visualization queries, such as filtering and grouping.
- A method for providing a uniform convergence rate for all `GROUP BY` categories, regardless of data membership in each group.
- An application of common interaction techniques to view and adjust sampling rates in progressive visualizations.

For the remainder of this paper, we will discuss relevant work in database and visual analysis systems. We will go into more detail on the original Wander Join algorithm, its limitations and our implementation and interface. We discuss our contributions, their methods and evaluation and finally our conclusion and future work.

2. Related Work

There is a clear need for maintaining highly responsive visual interfaces for exploring larger and larger databases. To achieve this, there has been recent interest in co-designing interactive visualization with the underlying data processing systems that compute the results rendered on the screen [4,5]. In this section, we review related work from both the visualization community, which focuses on visual analysis systems for large scale data, as well as the database community on scalable query processing.

2.1. Progressive Visual Analytics

Progressive Visual Analytics (PVA) techniques have become more common in the visualization community to address the challenge of big data. In a paper by Mühlbacher et al. [6], the authors described the interactions between the user and the automation along two dimensions: "direction of information" and "entity of interest". Although this organization does not uniquely apply to PVA but more broadly to all other types of automation, the paper suggests the means in which a user can affect

the outcomes of automation. Using this definition, our proposed Selective Wander Join approach falls under "Control" and "Execution" in that the purpose of Selective Wander Join is to provide the user with the ability to prioritize (or cancel) the execution of the queries.

More specific to PVA, Angelini et al. recently conducted a review of existing PVA techniques [7]. Similar to the paper by Mühlbacher et al., the work of Angelini et al. is not a survey, but a characterization of PVA techniques and a set of recommendations for future designs. Some commonalities between these techniques include the goal of alleviating the wait time in long or slow computation, the need to present to the user partial (incremental) results, and the design challenge of allowing users to provide feedback either with the intermediary results or the algorithm itself. However, while these design goals might be similar, the implementation and approach could sometimes differ significantly. Below we summarize some of the most relevant papers and techniques to our proposed system and discuss how Selective Wander Join differs.

The proposed Selective Wander Join aims at providing user guidance during the computation of expensive JOIN queries in databases. This topic differs from some other PVA approaches that focus on the steering of machine learning or data mining techniques, such as those by Stopler et al. on mining sequential data [8], Pezzotti et al. on steering t-SNE [9], and Turkay et al. on the designs of PVAs for high-dimensional data [10].

Focusing on PVA for database queries, systems such as sampleAction [11,12] directly employ online aggregation and interface concepts from the original CONTROL project [13] and evaluate their efficacy in real-world conditions. This system was instrumental in showing that users benefited from seeing immediate results alongside incrementally improving accuracies, and that users felt more empowered to actively explore their data instead of waiting for queries to complete. Their results also highlight the value of existing research in effective ways to visualize approximate and improving results. For example, Zgraggen et al. conducted a study to understand the effect of latency and blocking on user exploration [14]. The authors find that instantaneous results or results delivered in a progressive manner lead to a user discovering more insights. Similarly, Moritz et al. report that analysts using their system that delivers instantaneous but partial results (using sampling) felt more confident with their analysis outcomes [15].

There has been recent research that focuses on the interaction techniques of PVA. For example, ProgressiveVis [16] provides a toolkit that lets users ask for more points in visible location of a scatterplot. Similarly, Rosenbaum et al. [17] allow the user to steer the visualization and prioritize data regions of interest. InsightsFeed [18] demonstrates interface controls and visualization augments for progressive visualization steering.

Our proposed Selective Wander Join builds on top of the existing research, but with an emphasis on user-steering of multi-way JOIN queries in databases. JOINs are common in data analysis but are expensive to compute. Leveraging the recently published Wander Join technique [3], Selective Wander Join allows the user to direct computation resources towards aspects of the JOIN operations that are more relevant or time-sensitive to the user's analysis goals.

2.2. Data Systems for Interactive Data Exploration

Prior to the rise of PVA, a common method for supporting responsive interactive visualizations was to precompute different types of partial results or data structures that can speed up query execution. Different classes of queries, such as grouping or filtering on different attributes, warrant different methods of precomputation.

Data cubes have been widely used in the recent visualization literature [19–23]. In short, they efficiently store the results of a carefully crafted aggregation query (as in the above SQL query) that can be used to answer similar aggregation queries. To illustrate, if we `JOIN` a table by the attributes `(region, month)` and store the `COUNT(*)` for each combination of `(region, month)` values, then we can use this table to more quickly compute the `COUNT` of any grouping or filtering operation over

`region`, `month`, or both. The key is that subsequent queries only access the grouping attributes in the data cube.

Unfortunately, due to the necessary size and cost of computing, the data cube increases exponentially as the number of data dimensions increases. In addition, the data cube restricts queries to referencing the predetermined grouping attributes. This is undesirable in modern data exploration where the analyst may not know the exact set of tables and attributes she wants to analyze up front. In these cases, the user would need to wait minutes or hours to re-build the appropriate data cubes for each new query.

A second popular approach in visualization is to use sampling to reduce the number of records that need to be read from billions or trillions to hundreds or thousands [24–29]. This approach is promising because the potential latency reduction is considerable. However, typical sampling systems still rely on offline preprocessing where these systems assume a set of predetermined queries that the analyst will use (i.e., known query workload) in order to build a representative sample of the original data [30,31]. Once the samples are computed, the analyst's queries are executed using the smaller, sampled data to speed up the computation. Clearly, this has similar limitations as data cubes. What is needed is an approach that, with minimal setup, can flexibly and quickly respond to analysis queries without a restriction that the queries conform to a predetermined template.

There are a number of other techniques that can be used to support visual exploration and analysis of big data. Researchers have also explored the use of predictive prefetching [32–35] and specialized databases such as column stores [36] and in-memory databases [37]. Additionally, systems such as ScalaR [34] use a combination of these techniques. For a more comprehensive review of these techniques and their use in visualization, refer to the survey by Godfrey et al. [38].

2.3. Online Aggregation, Ripple Join, and Wander Join

Although the classic approach to improve query response times is to precompute all possible results, this can require a long wait time before the user sees any result. To address this issue, there has been recent interest in *online aggregation* algorithms that do not require precomputation and instead select samples "online" when the user submits a new query via the visualization. Such algorithms can provide real-time (approximate) query results as the systems draw samples from the database while executing the query.

The goal of this work is to reduce the number of samples necessary to achieve a desired error bound at a given confidence level. Reducing this *sample complexity* potentially means that the number of input records that need to be read can be reduced by orders of magnitude as compared to fully reading the input tables.

Extending the original online aggregation algorithm by Hellerstein et al. [39], Ripple Join was designed specifically to optimize `JOIN` queries in online aggregation [40]. Although Ripple Join is an order of magnitude more efficient than Hellerstein's online aggregation algorithm, unfortunately, Ripple Join still requires sample sizes that can be impractical for the interactive needs of visualization applications. For a query that joins tables `A` and `B`, the Ripple Join algorithm samples records from each table independently, and then checks whether or not they satisfy the `JOIN` condition, but often the join of two randomly chosen records is highly unlikely to pass. Thus, the number of records that need to be sampled can be very large before the confidence interval converges to a satisfactory level and the approximate query result is sufficiently close to the ground truth.

The Wander Join algorithm was recently introduced to address this issue [3]. Instead of sampling randomly from each table, Wander Join samples randomly from table `A` and then chooses a record from table `B` that the record can join with. This method increases the convergence rate of the query, resulting in less wait time for the user to achieve a higher confidence estimate.

Although Wander Join is a significant improvement over prior techniques for executing `JOIN` queries, it is not directly suited for PVA. A key limitation is that it draws samples independently of the `WHERE` and `GROUP BY` filters in the query. This problem is worsened when few records are able to

satisfy the filters, such as when the user is interested in a small segment of the dataset, or when the groups in the `GROUP BY` query exhibit skew. Crucially, these two conditions are common in PVA.

Although Wander Join performs significantly faster than previous online aggregation algorithms, Wander Join does not support user steering in the online sampling process. The original online aggregation algorithm did propose the notion of *index striding* to allow all groups of a `GROUP BY` query to converge at the same rate, and also enable the user to change the sampling rate of each group. However, this required a priori knowledge of what groups would be queried, as proper indexing structures and clustering were needed to support this method of sampling.

We evaluated Wander Join in PVA settings where users want to dynamically filter on expensive `JOIN` queries, and extended its techniques to reduce the sample complexity for this class of query workloads. This extension of Wander Join, which we call Selective Wander Join, addresses the needs of PVA where users often select subsets of data resulting in executing expensive `JOIN` queries with highly selective filters.

3. Wander Join Algorithm

Wander Join represents the state of the art in online aggregation across `JOINs`, providing online aggregation support for computing summary statistics such as sums and counts. More importantly, it intelligently selects which records to join with by modeling the `JOINs` as a join graph and walking along edges to find valid joined data. Figure 1 shows an example join graph for joining tables A, B and C. Each row (e.g., a1, a2, ...) represents a record in one of the tables, and an edge between two rows means that the two records can be joined in a natural join. Wander Join selects one record in table A, then randomly selects a path to table B, and repeats for table C. These edges are maintained by indexes on the tables. Indexes are a common database method to speed up access to each record, however these indexes need to be precomputed when data is loaded into the database.

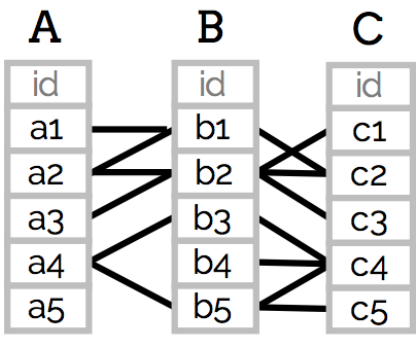

Figure 1. In Wander Join, `JOINs` are modeled as a graph, and random walks are taken along valid paths to select a sample.

An estimate of the aggregate value is returned after each walk. Since random walks will not return the uniform distribution needed to generate an unbiased estimate, Wander Join uses the *Horvitz-Thompson estimator*. This removes bias by dividing the value to be aggregated by the probability of having chosen the path taken to reach that value. For example, assuming the tables A, B and C can be joined across their common dimensions:

$$A(d1, d2) \bowtie B(d2, d3) \bowtie C(d3, d4) \tag{1}$$

We run a query to sum over the dimension $d4$. Wander Join selects a2 from the join graph below. It then randomly selects one of the edges leading from a2 to table B, resulting in either b1 or b2 as the next record. If it picks b2 then c1, c2 or c3 is chosen next according to the graph. The probability of selecting this path is $\frac{1}{5} \times \frac{1}{2} \times \frac{1}{3}$. Let $v(d4)$ be the value of the $d4$ attribute for this sample. The *Horvitz-Thompson estimator* would return: $v(d4)/(\frac{1}{5} \times \frac{1}{2} \times \frac{1}{3})$.

A walk will fail if a record does not pass a filter specified in the query. In this case, the aggregate value is treated as 0 to keep the *Horvitz-Thompson estimator* for SUM unbiased, since this sample is in the probability space of the distribution. However, this slows convergence of the estimate. For the rest of this paper, we treat the convergence rate as measuring the reduction of the estimate's relative standard error for the SUM aggregation:

$$\eta = \frac{z\frac{\sqrt{v}}{\sqrt{N}}}{E} \tag{2}$$

where v is the variance of the estimate, E, and N is the number of samples taken ($\frac{\sqrt{v}}{\sqrt{N}}$ is the standard error of the estimate). z is the z-score for the half width of the given confidence level. As the number of samples increases, the relative error approaches 0.

The relative error allows us to compare convergence rates over different queries.

3.1. Using Wander Join in Visual Exploration

In order to evaluate the effect our changes to Wander Join made, we reimplemented the Wander Join algorithm in Python and compared its performance to Selective Wander Join. To ensure that the comparison is fair, we: (1) evaluated the performance of the two algorithms based on the number of samples needed to reach convergence instead of clock time where language, experimental platform, etc. can affect the results, (2) used the same experiments described in the original Wander Join paper, and (3) extended the evaluation to include queries relevant to PVA where filtering using the `WHERE` and the `GROUP BY` clauses are common.

3.1.1. Data

Based on the original Wander Join paper, we evaluated the performance of Wander Join using Transaction Processing Performance Council Benchmark H (TPC-H), a synthetically generated dataset that simulates a data warehouse, and includes a set of queries that represent common analysis queries by business analysts. For our evaluation, we used four sizes of TPC-H data: 1 MB, 10 MB, 100 MB, and 2 GB. However, we found that regardless of the data size, the performance profile remains the same, likely due to the nature that TPC-H data is generated by drawing from an even distribution. As a result, we only report the evaluation results using the 10 MB dataset to reduce the impact of disk access delays during testing.

3.1.2. Validation Experiment

To verify that the accuracy of our Wander Join implementation is consistent with the original, we ran a validation experiment comparing performances for the following query to reach 95% confidence level:

```
SELECT sum(l_quantity)
FROM part, lineitem
WHERE part.p_partkey = lineitem.l_partkey
```

Figure 2 shows the number of samples needed to achieve various relative error values for the two implementations. The results of this experiment show that both implementations of Wander Join required the same number of samples to achieve the same confidence interval, thus validating that our implementation is faithful to the original.

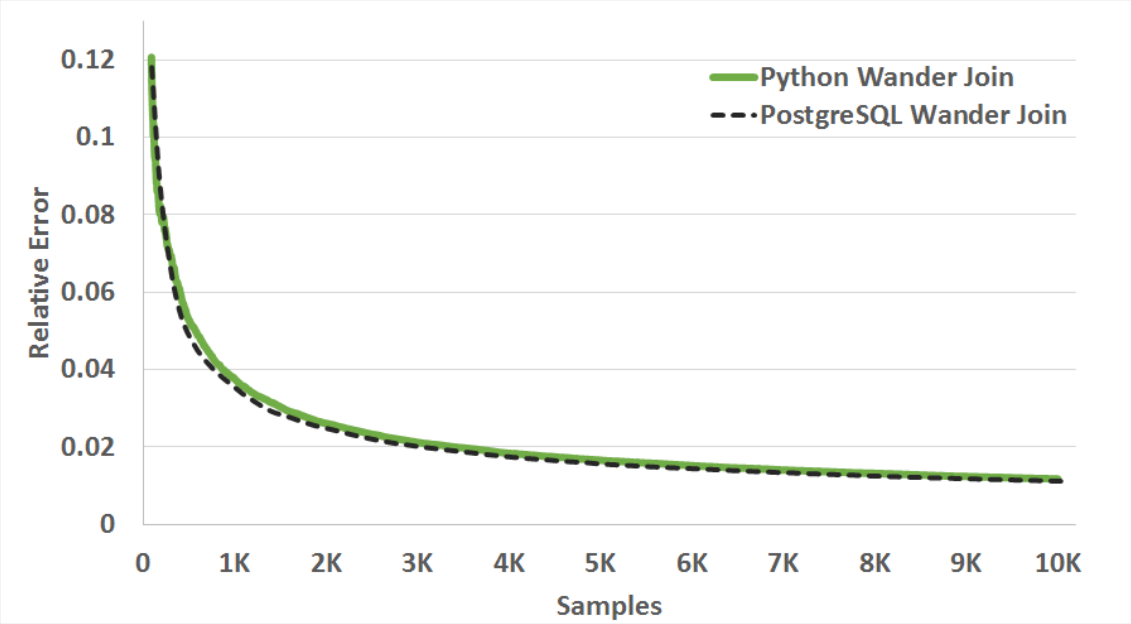

Figure 2. The relative error at 95% confidence level for our Python implementation of Wander Join and the original PostgreSQL implementation of Wander Join for 1 to 10,000 samples. Our reimplementation of the algorithm converges at the same rate as original Wander Join for a `JOIN` query with no filtering or `GROUP BY` in the query.

3.1.3. Evaluating `GROUP BY` Queries

We first evaluated Wander Join's performance for `GROUP BY` queries. We limited the dataset to have 40 different part sizes:

```
SELECT sum(l_quantity)
FROM part, lineitem
WHERE part.p_partkey = lineitem.l_partkey
GROUP BY part.p_size
```

This resulted in 40 groups, with an even distribution of the 60,175 records across all groups. It took approximately 25,000 total samples for each group to reach a maximum of 0.05 relative error. However, there are many situations where the records are not evenly distributed across all groups, such as time based events, or different categories where some categories are more popular than others. These types of datasets are common in the real world and often used in the visualization community.

Figure 3 compares the number of samples needed for all groups to converge to the same relative error for both the uniformly distributed and skewed datasets. It now takes Wander Join over 50,000 samples to achieve the same 0.05 relative error across all groups, nearly double the number of samples for an evenly distributed dataset. The group with 22% of the data is sampled more often than the other groups and reaches 0.05 relative error first. However, it is still sampled while the other groups converge, reducing the rate of convergence for the remaining groups, but also improving the relative error of the dominant group. The group with 22% of the data reaches 0.01 relative error while the other groups reach 0.05. This is 5 times higher in error of the estimate and a user cannot accurately compare the group estimates.

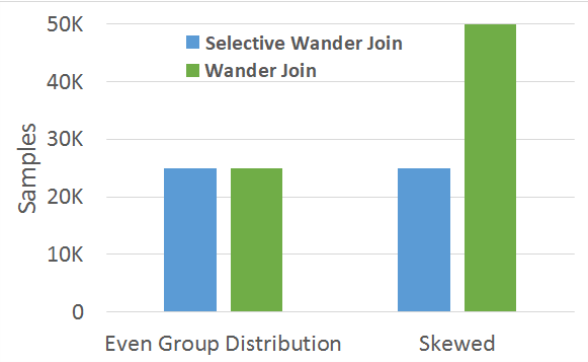

Figure 3. The number of samples needed for Wander Join and Selective Wander Join to achieve 0.05 relative error for all 40 groups in a `GROUP BY` query. Selective Wander Join and Wander Join require the same number of samples when the data is evenly distributed across all groups. However, Wander Join requires twice as many samples when 22% of data falls into 1 of the 40 groups. The more skewed the distribution is, the more samples Wander Join needs. Selective Wander Join samples evenly from all groups regardless of the distribution.

3.1.4. Evaluating Wander Join Using a Real World Dataset

The previous evaluation of Wander Join relies on the synthetic TPC-H dataset in accordance to the evaluation shown in the original Wander Join paper [3]. While TPC-H has been widely used in evaluating database systems, it is not representative of common data exploration tasks using visualization systems. In this section, we present an evaluation of Wander Join using the ASA Data Expo 09 [41] (commonly referred to as the "flight dataset" in the visualization community [20,22]). This dataset consists of a table with over 7 million records of commercial airline arrivals and departures across the United States in 2008 and an auxiliary data table with nearly 4500 rows of plane information. The total size of these two tables is approximately 675 MB.

Using the flight dataset, we evaluated Wander Join's performance on *unevenly* distributed data. Our query was to find the total number of flights per plane engine type:

```
SELECT count(*)
FROM plane_data, flights
WHERE plane_data.tail_num = flights.tail_num
GROUP BY plane_data.engine_type
```

This required a join between the flights and plane information tables and resulted in a highly skewed distribution. Turbo-Fan and Turbo-Jet were the most common engine type, with 68% and 27% of the flights respectively. Turbo-Prop had 4% of the flights while four other engine types all were less than 1% of the remaining flights.

With this dataset, when the most common engine type (Turbo-Fan) reaches 0.01 relative error, the four least common engine types are still above 0.1 relative error, while another engine type (Turbo-Prop) had relative error above 0.04. This high discrepancy in error of the estimate prevents a user from accurately comparing the group estimates. These relative errors are reached after approximately 11,000 samples. However, waiting for all groups to achieve 0.05 relative error would require over 977,000 samples, nearly two orders of magnitude more samples and time spent waiting before being able to compare group estimates.

3.1.5. Evaluating Selective Queries

In further testing, we evaluated the performance of Wander Join with queries involving `WHERE` clauses:

```
SELECT sum(l_quantity)
FROM part, lineitem
WHERE part.p_partkey = lineitem.l_partkey
```

```
AND part.p_size <= X
AND lineitem.l_quantity <= Y
```

This query returns the number of total parts sold for parts with sizes smaller than or equal to X and when the number sold per order is Y or less. X and Y were varied to filter out 0% to 99% of all rows in the full `JOIN`. If a filter is highly selective, most records do not pass the filter conditions (meaning the selectivity is the percent of rows that pass the filter conditions compared to the full `JOIN` without filtering).

Figure 4 shows the relative error for each level of filtering after 10,000 samples were taken for Wander Join. Note that when the selectivity is 10% (i.e., a *"highly selective"* filter because the `WHERE` clause in the query filters out 90% of the data), the uniform sampling approach of Wander Join can be inefficient. This is because most of the samples drawn by Wander Join in this highly selective query will be considered a "failed walk" as these samples do not satisfy the `WHERE` condition.

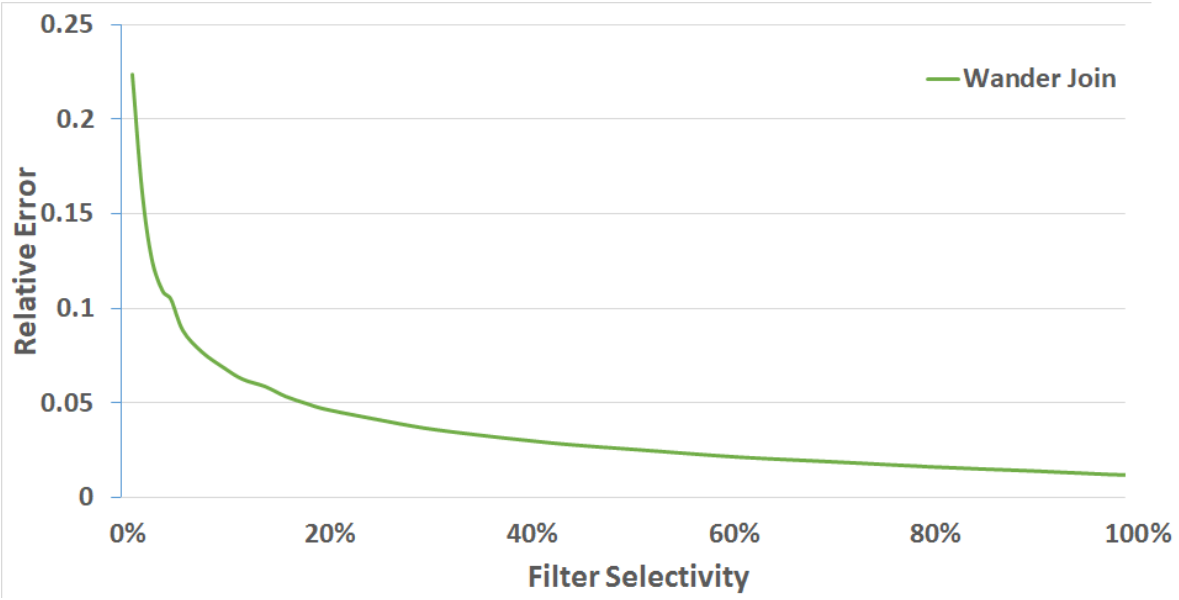

Figure 4. The relative error at 95% confidence level of Wander Join after 10,000 samples over varying levels of selectivity for the TPC-H dataset with scale factor of 0.01 and 60,175 rows in the full `JOIN`. The selectivity is the percent of records in the full `JOIN` that pass the filter conditions.The lower the selectivity percentage is, the slower the rate of convergence.

For Wander Join to achieve 0.01 relative error in this highly selective query, 208,000 samples would be needed. In contrast, the full `JOIN` of this query requires 60,175 rows. It is clear that in this case, the effectiveness of Wander Join is not only significantly reduced but Wander Join is over 3 times slower than running the full `JOIN`.

4. Limitations of Wander Join

The above experiments demonstrate both the potential of Wander Join and its limitations. Wander Join can support fast, iterative queries that do not require pre-computation and storage. This gives it great potential for use in progressive analytics. Further, it supports `JOIN`s that allow for flexible data analysis. In sum, Wander Join represents the state-of-the-art in online aggregation techniques.

However, Wander Join also has its limitations. While Wander Join is designed to be agnostic to the front-end visualization, it is a "black-box" algorithm that doesn't allow a user to direct or interact with the sampling process. We propose that this is a missed opportunity because sampling in Wander Join is inherently iterative and online. As the user sees the continuing outputs of Wander Join, it could aid the user's analysis process if she has the ability to "guide" Wander Join to sample from different

parts of a skewed dataset, thereby getting answers more quickly in areas of the visualization that she is more interested in.

Also, as shown in the experiments above, in cases where the query involves highly selective filters (with the use of the `WHERE` clauses) or when the data is unevenly distributed between groups in a `GROUP BY` clause, Wander Join can suffer from a performance point of view.

Unfortunately, since these highly selective filters and unevenly distributed datasets are common in visualization applications, extending Wander Join is necessary before it can be readily adopted by visualization researchers and practitioners.

5. Selective Wander Join: Wander Join for Visual Data Exploration

To address the limitations described above, we extended Wander Join to develop Selective Wander Join. The key algorithmic insight in Selective Wander Join is to take the `WHERE` and `GROUP BY` clauses into account when drawing samples from the database by prioritizing samples that are more likely to satisfy the filters. We achieve this by integrating the idea of importance sampling [2] into Wander Join, which allows us to prioritize samples by weighting them based on their importance to the query. Wander Join samples uniformly (the weight of each record is the same for all records), which can degrade the convergence rate under filtering and `GROUP BY` queries. By adding importance sampling, we can sample non-uniformly by changing record weights.

Importance sampling allows the user to influence the sampling process. It may be that the user does not want all groups to converge at the same rate. She is interested in seeing high confidence results from some of the groups more quickly than others. In this case, she can change the sampling rate of each group, and therefore increase or decrease the convergence rate for the targeted group. We designed an interface that will allow the user to see the sampling ratio of all groups relative to one another, and change the sampling rate by adjusting the weight applied to each group (Discussed in Section 6).

We can also apply importance sampling to uniformly sample from all groups in a `GROUP BY` query, resulting in all groups converging at the same rate, regardless of the number of records in each group. This is useful in progressive visualizations where users need to make visual comparisons between groups and different convergence rates can prevent the user from making accurate comparisons. We weight each record based on the number of records that also fall into that record's group and start sampling from the table containing the `GROUP BY` attribute. This allows Selective Wander Join to uniformly sample based on *group* instead of uniformly by *record* and all groups converge at the same rate. Now the user is not waiting for a group with low membership to converge.

We also apply importance sampling for filter queries to reduce the number of overall samples needed. For filter queries, we set the weight to 0 for records that do not pass the filters. This prevents Selective Wander Join from sampling the record again in the future, thereby reducing sample failures and the overall number of samples needed to reach convergence. All other records are sampled from uniformly. We do this online while sampling and update a record's weight to 0 if it does not pass the filter.

We implemented Selective Wander Join by extending our Python Wander Join implementation to use importance sampling. The source code for our implementation of Wander Join, Selective Wander Join and the evaluations are publicly available on Github (https://github.com/promarand/SelectiveWanderJoin), as well as the data used in our evaluations.

The following sections discuss how we applied importance sampling in more detail, as well as our evaluation of the methods and comparison to Wander Join.

5.1. Optimizing for Group By Queries

As discussed in Section 3.1.3, `GROUP BY` queries are ubiquitous in exploratory analysis. More often than not, the data rendered by a bar chart or heat map is computed as the result of a `GROUP BY` query over the x, or x and y, axis attributes.

Since Wander Join uniformly samples each record from the underlying table, each group's convergence rate depends on the proportion of records that belong to the group. Thus, it can take a large number of samples before the algorithm draws a record for an unpopular group. This prevents the user from making comparisons between groups during the online aggregation process, negating the benefits of progressively updating estimates.

5.1.1. Method

Our approach to optimizing for `GROUP BY` queries is to use importance sampling to uniformly sample *from each group* to ensure an uniform convergence rate. We achieve this desired outcome by weighting the records in each group relative to the number of records in the group and the number of total groups. Specifically, we set the weight of each record in the table referenced in the `GROUP BY` clause as:

$$\omega_i = \frac{1}{\alpha\beta} \tag{3}$$

where α is the number of records that are in the same group as record i (as calculated when the index is updated) and β is number of distinct groups. The intuition is that we want to sample from each group evenly, and sample each record within each group evenly. The $\frac{1}{\alpha}$ term ensures uniform sampling of records in a given group, while $\frac{1}{\beta}$ ensures uniformly sampling from each group.

With this weighting, Selective Wander Join randomly selects from the `GROUP BY` table first, guaranteeing even sampling from each group. The records in the next tables are weighted uniformly as before. Using Figure 1 as an example, assume table A contains the attribute we will group on and records $a_1 ... a_n$ will be weighted according to Equation (3). Records in table B and table C will be weighted uniformly. We do not need to adjust their sampling rates since we have already guaranteed even sampling by group from reweighting records in table A.

5.1.2. Evaluation

We ran the same `GROUP BY` query as in Section 3.1.3 on the evenly distributed TPC-H dataset, where equal number of records fell into each group. As expected, Selective Wander Join and Wander Join required the same number of samples to reach 0.05 relative error across all groups.

However, Selective Wander Join's advantages lie in unevenly distributed datasets. We also tested with the same flight dataset and query as in Section 3.1.4, where there is an uneven distribution of plane engine types across flights. Selective Wander Join only required 10,000 samples out of the 7 million records in order for all engine types to reach 0.05 relative error. Wander Join required over 975,000 samples for all groups to reach 0.05 relative error.

We compared the number of samples Selective Wander Join and Wander Join needed for all groups to reach 0.1 to 0.05 relative error, in 0.01 increments. As shown in Table 1, for each of these tests, Selective Wander Join only required 1% of the number of samples as Wander Join for the engine type query—resulting in a significant increase in performance on these queries.

Table 1. The number of samples needed by Selective Wander Join and Wander Join for all groups to achieve the same relative error rate for the *flights* `GROUP BY` query.

Relative Error	Selective Wander Join	Wander Join	Sample Ratio
0.05	9982	977,432	1.021%
0.06	3928	579,110	0.678%
0.07	3010	439,061	0.685%
0.08	1836	321,732	0.571%
0.09	1725	238,492	0.723%
0.10	1351	221,116	0.611%

5.2. Optimizing for Highly Selective Queries

Filtering data is a common task of exploratory analysis [42]. Consider a retail company business analyst interested in the average a customer spends on clothing, as opposed to all the items her company sells. Her query uses a filter to limit the average calculation to only include clothing items. This filter affects the *selectivity* of the query: the more restrictive the filter is, the more data is filtered out, resulting in a highly selective query. As seen in Section 3.1.5, Wander Join's performance in this scenario can be worse than executing the exact query.

5.2.1. Method

Conceptually, our approach to optimize Wander Join for highly selective queries is to prune out samples that do not pass the filter. By eliminating those records, we prevent sample failures and can converge with less samples (and therefore faster) than Wander Join.

In practice, when a filtering query is issued, we set the weight of each record uniformly. As we sample, if we encounter a record that fails the filter, we set the record's weight to 0 and consider this sample a failure. Although we still have a failure in the same sense as Wander Join, the key difference is that by setting the weight to 0, we guarantee that we never sample this failed record again. This is beneficial since Wander Join's sampling is with replacement; any record can be sampled multiple times, regardless of whether the record has failed the filter criteria once before. The bigger benefit is that this record is pruned from *all* paths that lead to it. Using Figure 5 as an example, we see that two possible paths to sample are a1→b1→c2 and a1→b2→c2. If Selective Wander Join selects a1, then b1, then c2 and record c2 fails the filter, c2's weight is now 0 and is pruned. Now not only has a1→b1→c2 been eliminated as a possible path, but also a1→b2→c2. By pruning out the failed record, we prevent any path from sampling that record again.

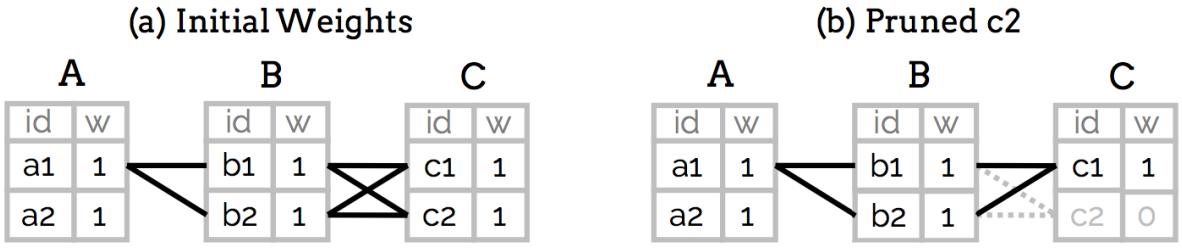

Figure 5. (**a**) A `JOIN` graph for a 3 table `JOIN`. Each record starts with a weight of 1. If Selective Wander Join chooses the path a1→b1→c2 and c2 fails the filter, c2 is pruned. (**b**) The resulting `JOIN` graph after pruning. Note that the path a1→b2→c2 has been pruned even though we never selected that path. Pruning out c2 prevents any path from sampling c2 again for this query.

5.2.2. Evaluation

We ran the same filtering queries as in Section 3.1.5, varied the *part.p_size* and *lineitem.l_quantity* filter to achieve levels of 0% to 99% filtering and used the same 10MB TPC-H dataset:

```
SELECT sum(l_quantity)
FROM part, lineitem
WHERE part.p_partkey = lineitem.l_partkey
AND part.p_size <= X
AND lineitem.l_quantity <= Y
```

Figure 6 captures the results of the selective filter queries. The lower the selectivity percentage, the larger improvement over Wander Join. For the 1% selective filter query, Selective Wander Join achieved 0.08 relative error after 10,000 samples, while Wander Join had 0.22 relative error. In terms of number of samples, Wander Join needed 65,000 to reach 0.08 relative error. Selective Wander Join is an

improvement over Wander Join by reducing the sample complexity by 85% (and there by speeding up convergence by a factor of 6).

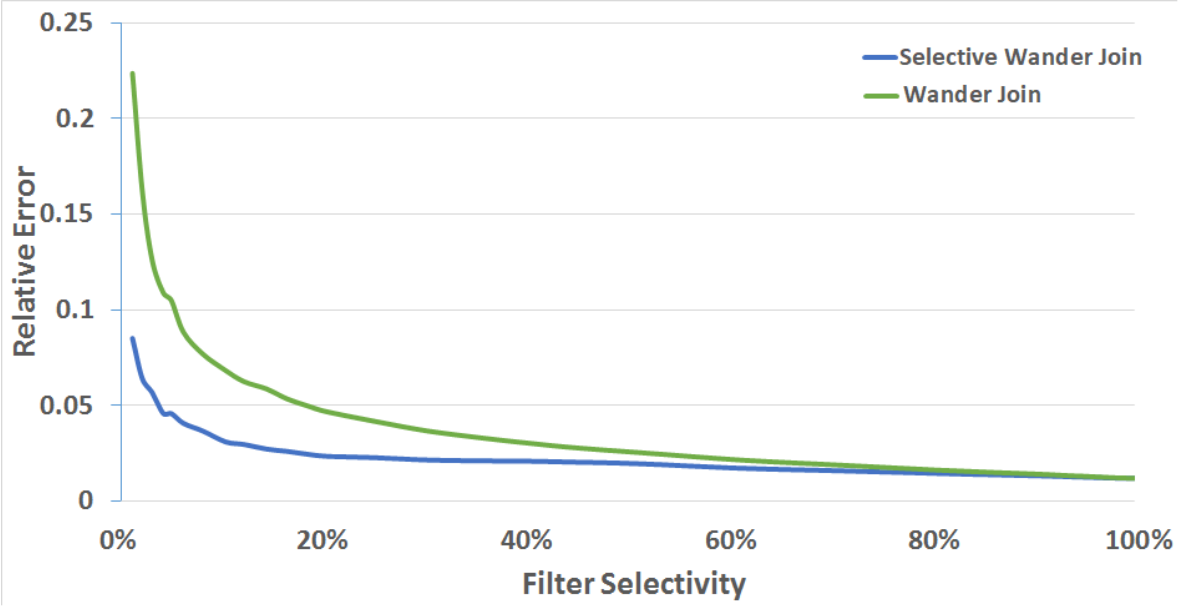

Figure 6. The relative error at 95% CI for both systems after 10,000 samples over various levels of selectivity for the TPC-H dataset. The lower the selectivity percentage, the slower the rate of convergence. By pruning out records that fail the filter, our relative error is less than standard Wander Join at 1% selective filter.

Selective Wander Join outperformed Wander Join at all selective filter queries and performed the same at the 100% selective filter query (since no records were filtered out). Although not shown, the filters can be on single or multiple values, as well as attribute values. When the filter condition is dependent on an attribute or involving a range, this makes prepruning via an index difficult.

Since online pruning requires sampling a record in order to prune it, one concern may be that we won't sample the same path multiple times if our data is much larger than the number of samples being taken. However, we did see that the same path can be selected multiple times in this case, and pruning can prevent this. We tested a query with a 99% selective filter on the TPC-H data scaled to 1 GB. This resulted in 6 million possible paths. We sampled 10,000 times and saw 117 different failed paths that were chosen twice or more. Pruning would have prevented these multiple failures. Additionally, we ran the 99% selective filter query to 0.08 relative error on both Wander Join and Selective Wander Join. It took Selective Wander Join 5000 (or 10%) fewer samples than Wander Join to achieve this relative error.

5.2.3. Extensions

A natural extension of online pruning would be to preprune out any records that do not pass the filters before starting sampling. This will prevent any failures from occurring and greatly increase the convergence rate. We performed an initial test of this idea on the 1 GB TPC-H dataset, and 99% selective filter query, meaning the `WHERE` clause filtered out 99% of the data. This resulted in 345 samples needed to reach 0.08 relative error. For online pruning it took 56,000 samples due to the high number of first failures.

This concept will be beneficial as long as the time required to preprune is insignificant over sampling time. Further work into effective methods of prepruning and evaluation are needed.

5.3. Trading Complexity for Usability

Progressive visualization systems require additional complexity beyond that of the underlying approximate query processing algorithm supporting them. Progressive visualization systems need to process and render query results in a human interpretable form. Interaction in the visualization system also increases algorithmic complexity.

Selective Wander Join extends Wander Join that optimizes sampling for visual analysis tasks and enables user interaction in the sampling process. This requires an increase in memory and computation time over that of the original Wander Join implementation. However, this increase in complexity allows for more effective use of the Wander Join algorithm in progressive visualization and the overall time increase may be offset by the reduced number of samples needed in the overall query execution.

Since we're keeping track of weights for each record in the database, there is additional space and time costs to consider for Selective Wander Join beyond that of the original Wander Join implementation. Additional storage is need to maintain the weight for each row being used in the query. However, since the weight is either 1 or 0 for filtering queries, only one bit of storage is needed. For a dataset with one billion rows, we will only need 125 MB. However, we also need to store the row index along with the weight. Assuming 1 billion rows again, each index can be stored in 30 bits. The total memory needed would be less than 4 GB, which can easily fit in memory in today's consumer systems.

Group By queries require additional memory. The first table in a group by query uses a fractional weight, while the remaining tables use binary weights (1 or 0). If we allow 32 bits to store the fractional weight, the total space needed to store the row index and weight for the first table in a group by query would be less than 8 GB in a table with 1 billion rows. The remaining tables in the Group By query require the same space as a table in a Filter query if the Group By query also includes filters. Otherwise, no weights need to be stored and rows are sampled uniformly.

Table 2 shows the amount of memory needed to store the weights and row indexes for different sized tables in filtering or grouping queries.

Table 2. The amount of memory needed to store the weights and row indexes per table of various table sizes. This assumes 31 bits per row index and 1 bit for a binary weight. The first table sampled in a Group By query with a join requires more bits to store the fractional weights. The remaining tables in the Group By query require the same space as a table in a Filter query if the Group By query also includes filters. Otherwise, no weights need to be stored and rows are sampled uniformly.

Table Size (Rows)	Filter Query	Group By Query (First Table)
10 k	40 kB	80 kB
100 k	400 kB	800 kB
1 M	4 MB	8 MB
10 M	40 MB	80 MB
100 M	400 MB	800 MB
1 B	4 GB	8 GB

Although there is additional memory and bookkeeping required, this increase in complexity allows for effective progressive visualization systems. Progressive visualization systems will trade off the decrease in data access speed to allow just the filtered data to be returned and allow for user-driven sampling.

6. User-Driven Sampling in Selective Wander Join

In addition to the performance improvement of Selective Wander Join over the original algorithm in both selective and `GROUP BY` queries, we note an additional benefit of Selective Wander Join when used in progressive visualization systems. Typical online aggregation algorithms (including the original online aggregation algorithm [39], Ripple Join [40], and Wander Join [3]) for progressive

visualizations are non-parametric in that they are executed in the background and the users cannot manipulate the behavior of the algorithms. Our proposed Selective Wander Join algorithm can function in the same way. However, in addition, it affords a "free parameter" that allows the user to interactively guide and focus the sampling process towards specific parts of the query that are most pertinent to the user's interest that can further speedup the execution of the query.

Specifically, in Selective Wander Join, by applying importance sampling, the user can readjust the weights so that groups of interest are sampled more often. This allows the estimate to converge faster while also allowing uniform sampling of the remaining groups. Additionally, groups of even less interest can be weighted to be sampled less often, or not at all if the weight is set to 0.

Selective Wander Join offers fine grained control of the sampling process for groups and provides interaction techniques that are unique to the system. The goal of our interface design is to demonstrate the ease of integrating Selective Wander Join into existing systems with established progressive visualization concepts. In Figure 7, we show a traditional progressive visualization where uncertainty is encoded as error bars and transparency. Weights for each group are also encoded on the visualization with a black bar as a percentage from 0 to 100%. The higher the percentage, the higher the sampling rate.

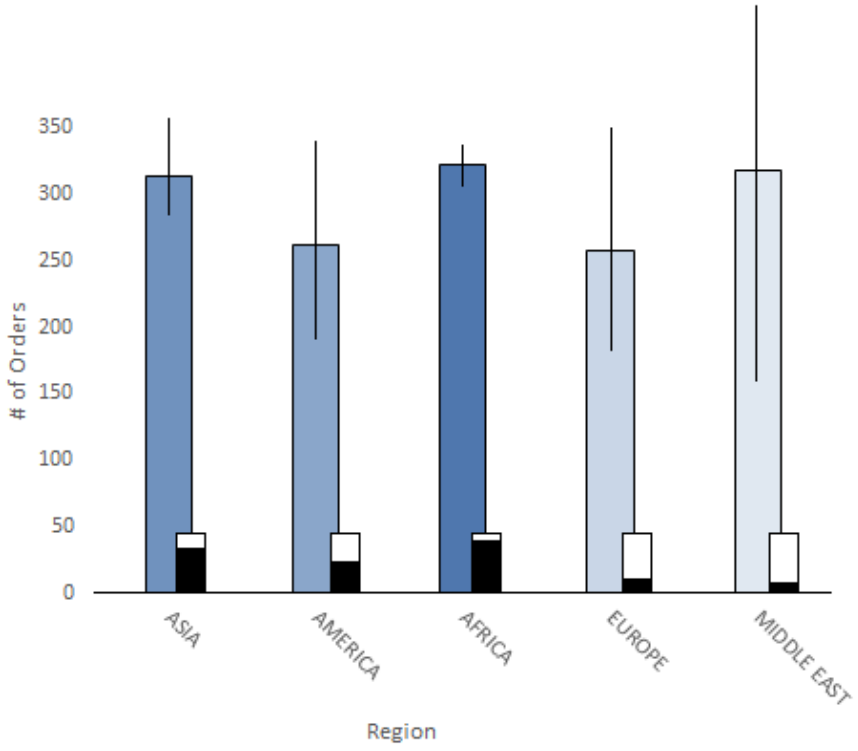

Figure 7. Demonstrating uncertainty encodings of relative error for bar charts using error bars and transparency. Weights for each group are encoded on the visualization with a black bar as a percentage from 0 to 100%. The higher the percentage, the heavier the weight (and therefore higher sampling rate).

In order for the user to adjust the weights of groups, we require a control on the visualization that displays the current weight and provides a method to adjust it. Figure 8a provides weight information and adjustment control via a vertical slider bar. The user can select a group of interest, which pops up a vertical slider bar showing the current weight of that group. The user can adjust the slider to increase or decrease the weight (and therefore the sampling rate) of the selected group.

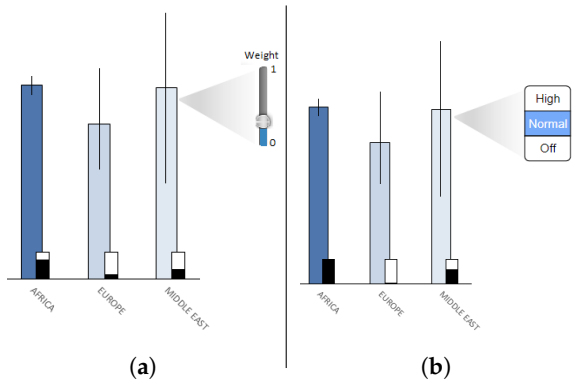

Figure 8. Two methods of controlling sampling rates in the progressive visualization. Our initial designs proposed a slider for fine granularity (**a**). Based on our findings from our user study, we refined the controls to be three buttons: High, Normal and Off (**b**). Users generally only used three positions of the slider and did not need fine grained control of sliders.

7. Expert User Study

We conducted a user study to observe how users would interact with user-driven sampling in Selective Wander Join, as well as receive feedback on our interface design and interaction controls.

7.1. Study Setup

We presented the system to three expert data analysts from a national research laboratory in separate evaluation sessions. All expert analysts were male and had 5 to 10 years of experience in data analysis. Their primary analysis tools are MATLAB and internally developed C++ applications to process and analyze sensor data. None use progressive visualizations or approximate values during analysis, as their current tools process data sequentially and then present results.

After a brief introduction and training on the use of Selective Wander Join, using the interface shown in Figure 9, we asked the participants to complete two tasks using the visualization to analyze a subset of the flight dataset. The participants were allowed as much time as they needed and were allowed to ask questions during the process. After the completion of the tasks, the participants answered a post-hoc questionnaire where they reported their findings and provided feedback about the system.

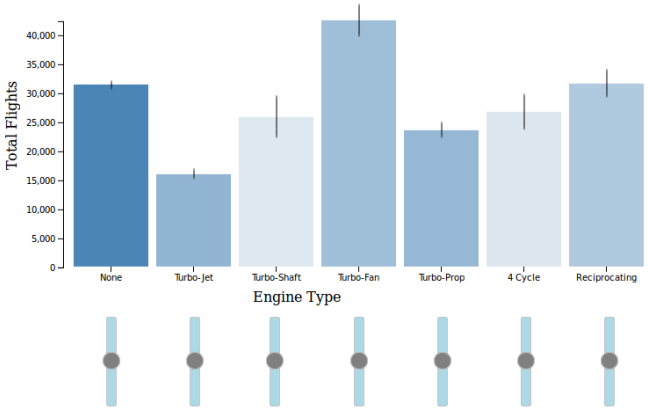

Figure 9. The interface used in the user study, to answer `GROUP BY` engine queries for the flight dataset.

Task 1 asked the analysts to find the engine type with the most number of flights. Task 2 asked the analysts to determine which of two given engine types had the fewer number of flights. The visualization is reset between tasks. The first task requires the users to evaluate all the groups in order to identify the engine type with the most flights. The second task encourages the users to focus

only on a subset of the groups. These two tasks allow us to contrast how users utilize the weights adjustments when there are groups of specific interest versus an exploratory comparative task.

The interface included a bar chart (shown in Figure 9) to support the query needed to complete both tasks, with the bar chart updating with new estimated values and relative error once per second. As described in the previous section, the transparency of the bars reflected the relative error of each group, and weight adjustments were made by a slider. The visualization was written in JavaScript and used the Python implementation of Selective Wander Join described in Section 5 for the back-end. Weight adjustments were sent to the Selective Wander Join back-end immediately and sampling rates were updated accordingly. No artificial delays were added beyond the latency in communication between the JavaScript front-end and Python back-end.

7.2. Results

7.2.1. Task 1 Results

Which engine type had the most number of flights? We found that all analysts were able to complete this task quickly as they answered correctly on their first attempt and prior to all groups reaching 0.01 relative error. As the relative errors across all groups began to reduce, we found that the analysts applied the weight adjustments in two different ways. Two analysts used the weight adjustment to narrow down the number of groups they needed to compare, by turning off the sampling for groups that clearly had too low of an estimate to be considered for the most number of flights. The third analyst instead left all the weights the same until he wanted to confirm the estimates of certain groups. In this case he increased the weight of the groups he was interested in as opposed to reducing the weight of the uninterested groups.

7.2.2. Task 2 Results

Between Turbo-Prop and Reciprocating, which engine type had the fewer number of flights? All analysts were able to complete this task quickly and easily by adjusting the weights upon start of the task. By phrasing the task as a comparison of specific groups, this ensured that the analysts were interested in only a subset of the available groups in the visualization. During this task, two of the analysts used the weight adjustment to decrease weights to 0 for all but the two specified groups. The third analyst increased the weight of the two specified groups instead of reducing the weight of the other groups.

7.3. Discussion

All analysts were able to complete the tasks without difficulty. They all adjusted the sliders to increase the sampling rate of interested groups. It is worth noting that a given analyst would use the weight adjustment in a similar way regardless of task. The same analysts that decreased weights to 0 in Task 1 also decreased weights to 0 in Task 2. The third analyst increased the weight of important groups in both tasks.

7.3.1. Efficacy of the Selective Wander Join Visual Interface

Overall, the analysts found the interface easy to use and the transparency for relative error helpful. One analyst remarked that they "waited for the color change" to be more confident in the estimate of that group and that he was "waiting for the darker blue to know it converged more".

The sliders served their purpose in offering weight adjustments, but may not have been the most efficient interaction technique if the user wanted all groups to return to the same sampling rate. One analyst said that once sliders were moved out of the center position, it was "difficult to recenter, and I would have to turn all of them up". This would set all groups to the maximum slider position and weight but still allow equal sampling rates from all groups.

7.3.2. Weight Adjustments

By observing how the analysts used the weight adjustment sliders and discussions with them after the tasks were complete, we found that users did not want or need the multiple levels of control over sampling groups that a slider provided. They either turned sampling for groups off or moved the sliders to the maximum position. Therefore we refined our interaction technique by replacing the sliders with a three button interface: High, Normal and Off. This new design is reflected in Figure 8b. For future work, we aim to formally evaluate the effectiveness of the simplified design versus the use of continuous sliders.

7.3.3. Timing of Weight Adjustments

We also noticed that the time when the analysts adjusted the weights would differ. For Task 2, when the analysts knew which groups would be relevant prior to issuing the query, we saw them immediately adjust the weights to prioritize those groups. In Task 1, they waited until they understood the general trends before adjusting weights. We also asked them specifically if they would prefer setting weights ahead of time or during the processing of the query. One of the analysts stated mid query adjustments were sufficient, as "the weights are easy to change once it has started". This also supports another analyst's statement that they would want the option to adjust the weights during the query, since they want a "first look, then adjust the weights".

8. Conclusions

We present Selective Wander Join, a progressive visualization system that extends online aggregation to enable interactive data exploration. Selective Wander Join improves on the latest online aggregation algorithm, Wander Join, by using importance sampling. This allows Selective Wander Join to converge faster on highly selective `WHERE` queries and converge uniformly on all groups in `GROUP BY` queries, regardless of data distribution. Selective Wander Join also provides a method for the user to adjust the convergence rate for each group, allowing users to prioritize groups of interest. We also presented interface designs and interaction techniques that would enable Selective Wander Join to integrate into a progressive visualization system.

We showed that Selective Wander Join outperforms Wander Join on filtering queries and performs equally as well on non-filtering queries. We also showed that Selective Wander Join requires up to 50% fewer samples than Wander Join to converge on all groups in a `GROUP BY` query.

We have only just scratched the surface on optimizing online aggregation for visual analysis. There are multiple opportunities for future work to explore. Adding preprocessing such as pruning out records that do not pass query filters would improve convergence rates even further. There are also other heuristics that can be applied to group sampling, such that importance sampling is not based solely on size, but on inherent stratification or the inherent skewness of the data. Additionally, there's the option to explore importance sampling as applied to rare or extreme values, so they are not missed in the sampling process. Additionally there may be optimizations in the calculation and storage of weights that can be explored.

There are other methods of sampling besides importance sampling that could be used in Selective Wander Join. One possible method is perceptual based sampling that leverages perception functions. Previous work in approximate query processing and perception [43,44] has shown this method can render approximate visualizations that are visually indiscernible from exact answer visualizations.

Overall, Selective Wander Join has been shown to be an improvement over current online aggregation methods for visualization tasks and through future work it can become an even more capable system.

Author Contributions: Conceptualization, M.P., C.S., E.W. and R.C.; Methodology, M.P., C.S., E.W. and R.C.; Software, M.P., C.S., E.W. and R.C.; Validation, M.P., C.S., E.W. and R.C.; Formal Analysis, M.P., C.S., E.W. and R.C.; Investigation, M.P., C.S., E.W. and R.C.; Resources, C.S., E.W. and R.C.; Data Curation, M.P., C.S., E.W. and R.C.; Writing—Original Draft Preparation, M.P., C.S., E.W. and R.C.; Writing—Review, Editing, M.P., C.S., E.W.

and R.C.; Visualization, M.P., C.S., E.W. and R.C.; Project Administration, R.C.; Funding Acquisition, C.S., E.W. and R.C.

Funding: This work was supported in part by National Science Foundation (NSF) 1527765, 1564049, 1513651, 1452977, and DARPA FA8750-17-2-0107.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Selinger, P.G.; Astrahan, M.M.; Chamberlin, D.D.; Lorie, R.A.; Price, T.G. Access path selection in a relational database management system. In Proceedings of the 1979 ACM SIGMOD international conference on Management of data, Boston, MA, USA, 30 May–1 June 1979; pp. 23–34.
2. Hastings, W.K. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika* **1970**, *57*, 97–109. [CrossRef]
3. Li, F.; Wu, B.; Yi, K.; Zhao, Z. Wander Join: Online Aggregation via Random Walks. In Proceedings of the 2016 International Conference on Management of Data, San Francisco, CA, USA, 26 June–1 July 2016; pp. 615–629.
4. Wu, E.; Battle, L.; Madden, S.R. The case for data visualization management systems: Vision paper. *Proc. VLDB Endow.* **2014**, *7*, 903–906. [CrossRef]
5. Wu, E.; Psallidas, F.; Miao, Z.; Zhang, H.; Rettig, L.; Wu, Y.; Sellam, T. Combining Design and Performance in a Data Visualization Management System. In Proceedings of the Conference on Innovative Data Systems Research, Chaminade, CA, USA, 8–11 January 2017.
6. Mühlbacher, T.; Piringer, H.; Gratzl, S.; Sedlmair, M.; Streit, M. Opening the black box: Strategies for increased user involvement in existing algorithm implementations. *IEEE Trans. Vis. Comput. Graph.* **2014**, *20*, 1643–1652. [CrossRef] [PubMed]
7. Angelini, M.; Santucci, G.; Schumann, H.; Schulz, H.J. A Review and Characterization of Progressive Visual Analytics. *Inform. Multidiscip. Dig. Publ. Inst.* **2018**, *5*, 31. [CrossRef]
8. Stolper, C.D.; Perer, A.; Gotz, D. Progressive visual analytics: User-driven visual exploration of in-progress analytics. *IEEE Trans. Vis. Comput. Graph.* **2014**, *20*, 1653–1662. [CrossRef] [PubMed]
9. Pezzotti, N.; Lelieveldt, B.; van der Maaten, L.; Hollt, T.; Eisemann, E.; Vilanova, A. Approximated and user steerable tsne for progressive visual analytics. *IEEE Trans. Vis. Comput. Graph.* **2016**, *23*, 1739–1752. [CrossRef] [PubMed]
10. Turkay, C.; Kaya, E.; Balcisoy, S.; Hauser, H. Designing Progressive and Interactive Analytics Processes for High-Dimensional Data Analysis. *IEEE Trans. Vis. Comput. Graph.* **2017**, *23*, 131–140. [CrossRef] [PubMed]
11. Fisher, D.; Popov, I.; Drucker, S. Trust me, I'm partially right: Incremental visualization lets analysts explore large datasets faster. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Austin, TX, USA, 5–10 May 2012; pp. 1673–1682.
12. Fisher, D. Incremental, approximate database queries and uncertainty for exploratory visualization. In Proceedings of the 2011 IEEE Symposium on Large Data Analysis and Visualization (LDAV), Providence, RI, USA, 23–24 October 2011; pp. 73–80.
13. Hellerstein, J.M.; Avnur, R.; Chou, A.; Hidber, C.; Olston, C.; Raman, V.; Roth, T.; Haas, P.J. Interactive data analysis: The control project. *Computer* **1999**, *32*, 51–59. [CrossRef]
14. Zgraggen, E.; Galakatos, A.; Crotty, A.; Fekete, J.D.; Kraska, T. How Progressive Visualizations Affect Exploratory Analysis. *IEEE Trans. Vis. Comput. Graph.* **2016**, *23*, 1977–1987. [CrossRef] [PubMed]
15. Moritz, D.; Fisher, D.; Ding, B.; Wang, C. Trust, but Verify: Optimistic Visualizations of Approximate Queries for Exploring Big Data. In Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, Denver, CO, USA, 6–11 May 2017.
16. Fekete, J.D. Progressivis: A toolkit for steerable progressive analytics and visualization. In Proceedings of the 1st Workshop on Data Systems for Interactive Analysis, Chicago, IL, USA, 17–21 October 2015; p. 5.
17. Rosenbaum, R.; Schumann, H. Progressive refinement: More than a means to overcome limited bandwidth. In Proceedings of the IS&T/SPIE Electronic Imaging, San Jose, CA, USA, 24 January 2009; p. 72430I.

18. Badam, S.K.; Elmqvist, N.; Fekete, J.D. Steering the craft: UI elements and visualizations for supporting progressive visual analytics. In *Computer Graphics Forum*; Wiley Online Library: Hoboken, NJ, USA, 2017; Volume 36, pp. 491–502.
19. Stolte, C.; Tang, D.; Hanrahan, P. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE Trans. Vis. Comput. Graph.* **2002**, *8*, 52–65. [CrossRef]
20. Lins, L.; Klosowski, J.T.; Scheidegger, C. Nanocubes for real-time exploration of spatiotemporal datasets. *IEEE Trans. Vis. Comput. Graph.* **2013**, *19*, 2456–2465. [CrossRef] [PubMed]
21. Liu, Z.; Jiang, B.; Heer, J. imMens: Real-time Visual Querying of Big Data. In *Computer Graphics Forum*; Wiley Online Library: Hoboken, NJ, USA, 2013; Volume 32, pp. 421–430.
22. Pahins, C.A.; Stephens, S.A.; Scheidegger, C.; Comba, J.L. Hashedcubes: Simple, low memory, real-time visual exploration of big data. *IEEE Trans. Vis. Comput. Graph.* **2017**, *23*, 671–680. [CrossRef] [PubMed]
23. Wang, Z.; Ferreira, N.; Wei, Y.; Bhaskar, A.S.; Scheidegger, C. Gaussian Cubes: Real-Time Modeling for Visual Exploration of Large Multidimensional Datasets. *IEEE Trans. Vis. Comput. Graph.* **2017**, *23*, 681–690. [CrossRef] [PubMed]
24. Agarwal, S.; Mozafari, B.; Panda, A.; Milner, H.; Madden, S.; Stoica, I. BlinkDB: Queries with bounded errors and bounded response times on very large data. In Proceedings of the 8th ACM European Conference on Computer Systems, Prague, Czech Republic, 15–17 April 2013; pp. 29–42.
25. Ding, B.; Huang, S.; Chaudhuri, S.; Chakrabarti, K.; Wang, C. Sample+ Seek: Approximating Aggregates with Distribution Precision Guarantee. In Proceedings of the 2016 International Conference on Management of Data, San Francisco, CA, USA, 26 June–1 July 2016; pp. 679–694.
26. Kamat, N.; Jayachandran, P.; Tunga, K.; Nandi, A. Distributed and interactive cube exploration. In Proceedings of the 2014 IEEE 30th International Conference on Data Engineering (ICDE), Chicago, IL, USA, 31 March–4 April 2014; pp. 472–483.
27. Li, X.; Han, J.; Yin, Z.; Lee, J.G.; Sun, Y. Sampling cube: A framework for statistical olap over sampling data. In Proceedings of the 2008 ACM SIGMOD international conference on Management of data, Vancouver, BC, Canada, 10–12 June 2008; pp. 779–790.
28. Fekete, J.D.; Primet, R. Progressive analytics: A computation paradigm for exploratory data analysis. *arXiv* **2016**, arXiv:1607.05162.
29. Im, J.F.; Villegas, F.G.; McGuffin, M.J. Visreduce: Fast and responsive incremental information visualization of large datasets. In Proceedings of the 2013 IEEE International Conference on Big Data, Santa Clara, CA, USA, 6–9 October 2013; pp. 25–32.
30. Chaudhuri, S.; Das, G.; Narasayya, V. Optimized stratified sampling for approximate query processing. *ACM Trans. Database Syst.* **2007**, *32*, 9. [CrossRef]
31. Park, Y.; Cafarella, M.; Mozafari, B. Visualization-aware sampling for very large databases. In Proceedings of the 2016 IEEE 32nd International Conference on Data Engineering (ICDE), Helsinki, Finland, 16–20 May 2016; pp. 755–766.
32. Doshi, P.R.; Geraldine, E.; Rosario, G.; Rundensteiner, E.; Ward, M. A strategy selection framework for adaptive prefetching in data visualization. In Proceedings of the 15th International Conference on Scientific and Statistical Database Management, Cambridge, MA, USA, 9–11 July 2003; pp. 107–116.
33. Chan, S.M.; Xiao, L.; Gerth, J.; Hanrahan, P. Maintaining interactivity while exploring massive time series. In Proceedings of the IEEE Symposium on Visual Analytics Science and Technology, Columbus, OH, USA, 19–24 October 2008; pp. 59–66.
34. Battle, L.; Chang, R.; Stonebraker, M. Dynamic prefetching of data tiles for interactive visualization. In Proceedings of the 2016 International Conference on Management of Data, San Francisco, CA, USA, 26 June–1 July 2016; pp. 1363–1375.
35. Cetintemel, U.; Cherniack, M.; DeBrabant, J.; Diao, Y.; Dimitriadou, K.; Kalinin, A.; Papaemmanouil, O.; Zdonik, S.B. Query Steering for Interactive Data Exploration. In Proceedings of the Conference on Innovative Data Systems Research (CIDR), Asilomar, CA, USA, 6–9 January 2013.
36. Stonebraker, M.; Abadi, D.J.; Batkin, A.; Chen, X.; Cherniack, M.; Ferreira, M.; Lau, E.; Lin, A.; Madden, S.; O'Neil, E.; et al. C-store: A column-oriented DBMS. In Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, 30 August–2 September 2005; pp. 553–564.

37. Kemper, A.; Neumann, T. HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots. In Proceedings of the 2011 IEEE 27th International Conference on Data Engineering (ICDE), Hannover, Germany, 11–16 April 2011; pp. 195–206.
38. Godfrey, P.; Gryz, J.; Lasek, P. Interactive visualization of large data sets. *IEEE Trans. Knowl. Data Eng.* **2016**, *28*, 2142–2157. [CrossRef]
39. Hellerstein, J.M.; Haas, P.J.; Wang, H.J. Online aggregation. *ACM SIGMOD Rec.* **1997**, *26*, 171–182. [CrossRef]
40. Haas, P.J.; Hellerstein, J.M. Ripple joins for online aggregation. *ACM SIGMOD Rec.* **1999**, *28*, 287–298. [CrossRef]
41. Wickham, H. ASA 2009 Data Expo. *J. Comput. Graph. Stat.* **2011**, *20*, 281–283. [CrossRef]
42. Shneiderman, B. The eyes have it: A task by data type taxonomy for information visualizations. In Proceedings of the IEEE Symposium on Visual Languages, Boulder, CO, USA, 3–6 September 1996; pp. 336–343.
43. Alabi, D.; Wu, E. PFunk-H: Approximate query processing using perceptual models. In Proceedings of the Workshop on Human-In-the-Loop Data Analytics, San Francisco, CA, USA, 26 June–1 July 2016; p. 10.
44. Wu, E.; Nandi, A. Towards Perception-aware Interactive Data Visualization Systems. In Proceedings of the DSIA Workshop, Chicago, IL, USA, 26 October 2015.

© 2019 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).