QAGView: Interactively Summarizing High-Valued Aggregate Query Answers

Yuhao Wen, Xiaodan Zhu, Sudeepa Roy, Jun Yang
Department of Computer Science, Duke University
Durham, North Carolina
ywen@cs.duke.edu,willzxd@gmail.com,{sudeepa,junyang}@cs.duke.edu

ABSTRACT

Methods for summarizing and diversifying query results have drawn significant attention recently, because they help present query results with lots of tuples to users in more informative ways. We present QAGView (Quick AGgregate View), which provides a holistic overview of high-valued aggregate query answers to the user in the form of *summaries* (showing high-level properties that emerge from subsets of answers) with *coverage* guarantee (for a user-specified number of top-valued answers) that is both *diverse* (avoiding overlapping or similar summaries) and *relevant* (focusing on high-valued aggregate answers). QAGView allows users to view the high-level summaries as clusters, and to expand individual clusters for their constituent result tuples. Users can fine-tune the behavior of QAGView by specifying a number of parameters according their preference. To help users choose appropriate parameters interactively, QAGView employ a suite of optimizations that enable quick preview of how the quality of the summaries changes over wide ranges of parameter settings, as well as real-time visualization of how the summaries evolve in response to parameter updates.

CCS CONCEPTS

• **Human-centered computing** → **Information visualization**;
• **Information systems** → *Clustering*; *Top-k retrieval in databases*;

KEYWORDS

Database visualization; database usability; clustering; data mining

ACM Reference Format:
Yuhao Wen, Xiaodan Zhu, Sudeepa Roy, Jun Yang. 2018. QAGView: Interactively Summarizing High-Valued Aggregate Query Answers. In *SIGMOD'18: 2018 International Conference on Management of Data, June 10–15, 2018, Houston, TX, USA*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3183713.3193566

1 INTRODUCTION

Summarization and diversification of query results have recently drawn significant attention in databases and other applications such as keyword search, recommendation systems, and online shopping. The goal of both result summarization and result diversification

Rank	hdec	age	gender	occupation	val
1	1975	20s	M	Student	4.24
2	1980	20s	M	Programmer	4.13
3	1980	10s	M	Student	3.96
4	1980	20s	M	Student	3.91
5	1985	20s	M	Programmer	3.86
6	1980	20s	M	Engineer	3.83
7	1985	10s	M	Student	3.77
8	1985	20s	M	Student	3.76
9	1995	30s	F	Educator	370
10	1985	20s	M	Engineer	3.65
. . . .					
41	1995	60s	M	Retired	3.09
42	1995	30s	M	Scientist	3.07
43	1995	30s	M	Marketing	3.02
44	1995	20s	M	Technician	2.92
45	1995	30s	M	Entertainment	2.91
46	1995	20s	M	Executive	2.91
47	1995	30s	F	Librarian	2.84
48	1995	30s	M	Student	2.81
49	1995	20s	M	Writer	2.51
50	1995	20s	F	Healthcare	1.98

Figure 1: Tuples in the result of the query in Example 1.1.

is to make a large set of query result tuples more informative to the user, since the user is unlikely to view results beyond a small number. In this demonstration, we present QAGView (Quick AGgregate View), which produces an "overview" of high-valued tuples in the result of an aggregate database query as a set of *clusters*, with compact and user-interpretable *summaries* of their constituent result tuples. QAGView helps users find the desired clustering that meet the following (often competing) goals simultaneously: 1) the clusters must provide good *coverage*, i.e., covering at least a specified number of top-valued result tuples; 2) they must be *diverse*, i.e., avoiding overlapping clusters and/or penalizing similar summaries; and 3) they must be *relevant*, i.e., avoiding dilution caused by covering low-valued result tuples.

Example 1.1. *Suppose an analyst is using the movie ratings data from* MovieLens *[2] to investigate average movie ratings across genres by different groups of users over time. The analyst first joins several relations from this dataset (information about movies, ratings, users, and their occupations) into one relation R, extracts some additional features from the original attributes (age group, decade, half-decade), and then runs the following aggregate query on R (we omit the join for simplicity). In this query,* hdec *denotes disjoint five-year windows of half-decades, e.g., 1990 for 1990–94, 1995 for 1995–99, etc.;* agegrp *denotes age groups of users, e.g., 10s for 10–19, 20s for 20–29, etc.*

```
SELECT hdec, agegrp, gender, occupation, avg(rating) as val
FROM R
GROUP BY hdec, agegrp, gender, occupation
WHERE genres_adventure = 1
HAVING count(*) > 50
ORDER BY score DESC
```

The top 10 and bottom 10 results from this query are shown in Figure 1. Suppose the analyst has no patience or interest to go over all

50 result tuples; instead, we get to deliver 4 rows of information that will best help the analyst understand viewers and time periods with high ratings for the adventure genre.

The obvious option is to output the top 4 result tuples from Figure 1, but they do not summarize the common properties of the intended viewers/times periods. In addition, despite having high scores, they have attribute values that are close to each other (e.g., male students in their 20s) leading to repetition of information and sub-optimal use of the designated space of 4 rows. More importantly, the top 4 original result tuples may give a wrong impression on the common properties of high-valued tuples even if they all share those properties. For instance, all top 4 tuples share the properties (20s, M), but it is misleading, since a closer look at Figure 1 reveals that many tuples with low values (49th, 46th, 44th) share these properties too, suggesting that male viewers in their 20s may or may not give high rating to the adventure genre. A corollary to the above observation is that simply running a standard clustering algorithm on the high-valued result tuples may not work, because some resulting clusters may include low-valued result tuples and hence are misleading.

`QAGView` provides solutions to such scenarios and helps the user see summaries of high-valued result tuples with *relevance, coverage, and diversity*. Each summary cluster is described by a pattern that sets a subset of the group-by attributes to specific values while leaving others to "don't-cares" (denoted by $*$). Based on their preference, users use three parameters to control the behavior of `QAGView`. The *size parameter* k specifies the number of clusters desired ($k = 4$ in Example 1.1). The *coverage parameter* L denotes the number of top-valued result tuples (as ranked by the original aggregate query) that must be covered by the k clusters. The *diversity parameter* D ensures that the clusters should be at least "distance" D from each other; i.e., between any two cluster patterns, there are at least D attributes for which they do not set identical specific values.

Notably, `QAGView` presents three interfaces for users to explore and understand aggregate query results.

1. Exploring high-valued result tuples via summary clusters. Once `QAGView` summarizes the query result given k, L, D, it shows each summary cluster as a row; users can then click on the row to expand the cluster to reveal its constituent result tuples.

For example, suppose we run `QAGView` for the query in Example 1.1 with $k = 4, L = 8$, and $D = 2$; i.e., the user would like to see at most 4 clusters, these clusters should cover the top 8 tuples from Figure 1, and any two clusters should disagree on at least two attributes. `QAGView` would produce the four clusters shown in the top of Figure 2, with their patterns as well as average values of their constituent tuples. The user can investigate any of these clusters further by expanding them (clicking on ▼). The bottom of Figure 2 shows when all four clusters are expanded. Here, it happens that clusters are disjoint and cover no other tuples outside the top 8; in general, the clusters may overlap and contain lower-valued tuples, although `QAGView` will attempt to avoid such cases.

The above example illustrates several advantages and features of `QAGView` in providing a meaningful and holistic summary of high-valued aggregate query answers. *First,* the properties that summarize multiple top result tuples are clearly highlighted by the summary clusters, whereas the original top 8 result tuples are still accessible by expanding the clusters. *Second,* the summary clusters are diverse, each contributing some extra novelty to the answer. *Third,* the clustering captures the properties of the top result tuples that distinguish them from those with low values (e.g., (20s, M) is not chosen as discussed in Example 1.1), which cannot be achieved by applying standard clustering methods to the top L tuples.

hdec	agegrp	gender	occupation	avg val	
1975	**20s**	**M**	**Student**	**4.24**	▼
1980	*	**M**	*	**3.96**	▼
1985	**20**	**M**	**Programmer**	**3.86**	▼
1985	*	**M**	**Student**	**3.76**	▼

hdec	agegrp	gender	occupation	avg val	rank
1975	**20s**	**M**	**Student**	**4.24**	▼
1975	20s	M	Student	4.24	1
1980	*	**M**	*	**3.96**	▼
1980	20s	M	Programmer	4.13	2
1980	10s	M	Student	3.96	3
1980	20s	M	Student	3.91	4
1980	20s	M	Engineer	3.83	6
1985	**20**	**M**	**Programmer**	**3.86**	▼
1985	20s	M	Programmer	3.86	5
1985	*	**M**	**Student**	**3.76**	▼
1985	20s	M	Student	3.76	7
1985	10s	M	Student	3.76	8

Figure 2: Collapsed (top) vs. expanded (bottom) views of the summary clusters.

2. Guiding the selection of clustering parameters. Naturally, selecting the right values of k (number of clusters), L (tuples to cover), and D (distance to ensure diversity) is non-trivial, and highly dependent on how users intend to use the aggregate query result. Thus, instead of imposing automatically chosen parameter settings on users, `QAGView` aims to help users select appropriate settings via interactive exploration. For instance, there may be scenarios when users would like to find a certain parameter setting that produces a moderate number of well-separated clusters that do not include too many low-valued tuples. Finding such settings by repeatedly tweaking parameters can be time-consuming. `QAGView` allows users to quickly preview, as line plots, the effect of parameters on clustering quality over a wide range of parameter settings, so that users can visually and quickly identify specific settings of interest. For example, `QAGView` can produce the visualization in Figure 4 to show how clustering quality—as measured by the overall average value among all covered tuples (including those outside the top L)—changes with k and D.

3. Visualizing the evolution of clusters as parameters change. Besides producing the overall preview of how parameter settings affect clustering quality, `QAGView` also helps users visualize how clusters changes at a more fine-grained level between two parameter settings. Upon tweaking some parameters, users can visually compare the old versus clusters side by side. An example is shown in Figure 5. With this visualization, users can quickly get summary information about the clusters, see how result tuples get redistributed, and find out which clusters remain the same and which ones are merged, split, or reorganized.

Related work. There has been work on *diversifying* a set of result tuples by selecting a subset of them. For example, *diversified top-k* [4] takes into account diversity and relevance while selecting top result tuples. *DisC diversity* [1] considers similarity with the tuples that are not selected as well as and diversity and relevance among the selected ones. In contrast, we intend to produce *summarized* information on the result tuples to give the user a holistic view of the result tuples with high value. Another line of related work is *smart drill-down* [3], which explores and summarizes *interesting tuples* in a database (not aggregate answers) using a notion of diversity called "marginal counts." These previous approaches all consider different problem definitions and usage scenarios from ours; we refer interested readers to [5] for a detailed discussion.

2 QAGVIEW SYSTEM DETAILS

In this section we discuss some of the system details of `QAGView`. `QAGView` is a web-based application coded in Java, Scala, and JavaScript, based on Play Framework 2.6 with D3 for visualization and PostgreSQL as the database backend. In the following, we highlight some algorithmic and optimization challenges in supporting interactivity. For additional details, please see [5].

Computing summary clusters. Given an aggregate query Q on a database $\mathcal{D}$, where the query has m group-by attributes and each tuple t in the output $Q(\mathcal{D})$ has an aggregate value $\mathtt{val}(t)$ (score of the tuple, higher is better), `QAGView` outputs a set of (possibly overlapping) clusters O for the tuples in $Q(\mathcal{D})$, where each cluster is defined by a pattern that sets a subset of the m group-by attributes to specific values while leaving others to "don't-cares" (denoted by $*$). A feasible solution is a set of clusters O that satisfy the following requirements: 1) **size** k**:** the number of clusters in O is at most k; 2) **diversity** D**:** the *distance* $d(C_1, C_2)$ between any two clusters C_1, C_2 is at least D, i.e., C_1 and C_2's patterns cannot set identical *specific* (non-$*$) values for more than $m - D$ group-by attributes; 3) **coverage** L**:** the output clusters cover the original top-L tuples in $Q(\mathcal{D})$; and 4) **minimality:** no cluster in O should contain another in O. The goal is to find a solution with highest quality, defined as the average aggregate value among all result tuples covered by the solution: $\mathtt{val}(O) = \left(\sum_{t \in \mathtt{cov}(O)} \mathtt{val}(t)\right) / |\mathtt{cov}(O)|$. Here, $\mathtt{cov}(C)$ denotes the result tuples covered by cluster C, and $\mathtt{cov}(O) = \bigcup_{C \in O} \mathtt{cov}(C)$ are the tuples covered by all clusters in O. Note that $\mathtt{cov}(O)$ may contain tuples outside the desired top L tuples, potentially lowering the solution quality.

The optimization problem above is NP-hard even when some of the constraints involving k, L, D are relaxed [5]. We have designed efficient heuristic algorithms that exploit the *semilattice* structure of the clusters and its properties (e.g., merging two clusters does not violate the diversity constraint involving D). Two natural approaches are *bottom-up* (starting with the set of singleton clusters each containing an individual top-L tuple, and greedily merging clusters until all solution constraints are met), and *fixed-order* (starting with an empty solution, adding one singleton cluster containing an individual top-L tuple at a time in some order, and merging clusters greedily whenever solution constraints are violated). Through experiments, we have found that *bottom-up* tends to generate solutions with higher quality but *fixed-order* is better in terms of efficiency. So in `QAGView`, we take the advantages of both in a *hybrid* algorithm, which runs the faster *fixed-order* with a target number of clusters $k' > k$, and then switch to the more careful *bottom-up* (now with much fewer starting number of clusters) to obtain the final solution (details and pseudocodes in [5]).

Guiding the selection of clustering parameters. Users typically have some idea about how to set L (i.e., what aggregate values are consider high), but need help setting k and D. To produce visualizations such as the one in Figure 4, we must quickly compute, for a given L, clustering solutions for all possible combinations of k and D values. Independently computing a solution for each (k, D) pair would take too much time. Instead, we adapt the *hybrid* algorithm such that given L and D, with one execution, it can compute (and remember) solutions for all k values of interest. Briefly, given L and D, in the first phase of *hybrid*, we use *fixed-order* to compute a solution with a sufficiently large number of clusters. Then, in the second phase, we use *bottom-up* to decrease the number of clusters gradually by greedily merging clusters; each step of *bottom-up* produces a solution for a different k value. Instead of remembering the clusters in every solution, we note that each cluster always exists in the solutions for a range of consecutive k values. Hence, we can efficiently store and index all solutions by associating each cluster with a range of k values.

In our experiments [5], we have found that using the above method to precompute and index solutions for all k values takes less than twice as long as computing the solution for a single k value. On the other hand, it supports production of visualizations such as Figure 4 in real time. Furthermore, with the solution index it creates, a solution for any given k can be produced much faster (more than 200 times faster than computing it from scratch), which enables interactive user adjustments to clustering parameters.

Other optimizations to improve interaction speed. `QAGView` implements a number of other optimizations to ensure fast response time especially when challenged by large result sets.

Incremental evaluation of greedy objective. Each greedy step in our algorithms considers all possible cluster merges, which is expensive since the resulting solution quality needs to be computed for each possibility. Noting that changes to solution quality come from new result tuples not yet covered by existing clusters, we have devised a method that avoids recomputing a cluster's effect on solution quality from scratch in every greedy step, by remembering and incrementally computing this effect as needed. Our experiments show about 30× speed-up using this method.

Better cluster enumeration and matching. When the number of group-by attributes is high, simply enumerating all possible clusters would be impractical. Instead, we ensure that our algorithms consider only clusters that contain at least one top-L result tuple. Moreover, finding all result tuples covered by a given cluster is often more expensive than finding clusters covering a given result tuple, because the latter requires only examining the tuple's attribute values to generate cluster patterns. Optimizations based on these ideas bring another factor of 100–1000× speed-up.

Hashing strings to integers. In many datasets, a large fraction of attributes are strings; storing, manipulating, and matching these strings as part of the cluster patterns can create considerable overhead. Thus, we establish a hash-based mapping between strings and integers, allowing string-valued attributes to be internally processed efficiently as integer-value ones. This optimization brings about 50× speed-up.

Details of our experiments can be found in [5]. We evaluated our algorithms and optimizations on a large TPC-D dataset for scalability. With about 50k result tuples and $L = 2000$, `QAGView` takes about 3.5 seconds to precompute solutions for all (k, D) values of interest and generate the visualization to guide their selection; subsequently, viewing the solution for a specific (k, D) pair takes only a few milliseconds.

Visualizing cluster changes. As users adjust clustering parameters, QAGView produces a visualization like that in Figure 5 to help users understand how their adjustment changes an old clustering solution O_1 to a new one O_2. We display the clusters in O_1 and O_2 as two vertical stacks of rectangles, with width proportional to the number of result tuples they contain. For any two clusters $C_1 \in O_1$ and $C_2 \in O_2$ whose contents overlap, we further show a band from C_1 to C_2 with width (in the middle) proportional to the size of the overlap. How to present such relationships between clusters in the old and new solutions in a clean and informative manner is challenging. If we simply stack the clusters in an arbitrary order, lots of bands will cross, creating visual clutter. We approach this challenge in a principled way, formulating it as an optimization problem that looks for the best visual placement of clusters to minimize a measure of "clutter." We reduce this problem to bipartite graph matching, which can be solved efficiently in polynomial time. In practice, generating the optimized visualization takes only a few milliseconds in our experiments [5].

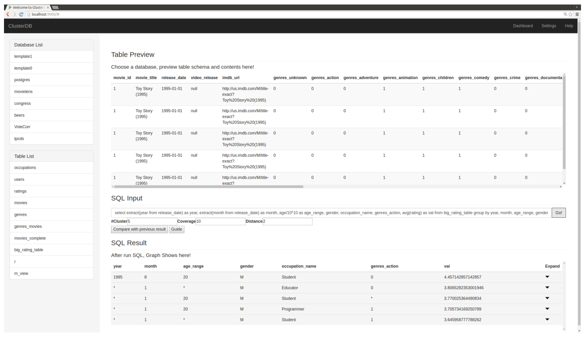
3 DEMONSTRATION

The demonstration of QAGView uses three datasets—besides the *MovieLens* data discussed in Example 1.1, we will also let users explore voting records of legislators in the U.S. Congress as well as player performance statistics in the National Basketball Association.

1. Exploring high-valued result tuples via summary clusters. As Figure 3 illustrates, from the main GUI of QAGView, users can select which database to connect to and what table to view. They can issue a SQL query and specify the clustering parameters k, L, and D (defaults are provided), and then explore the query result tuples as a list of summary clusters, each of which can be further expanded to reveal the list of result tuples therein.

2. Guiding the selection of clustering parameters. Users have the option of asking QAGView to "guide" them through the selection of clustering parameters. Given the coverage parameter L, QAGView plots how the number of clusters (k) affects the clustering quality under different settings of the diversity parameter (D), as illustrated in Figure 4. Each line plot (of different color) represents a different D. Intuitively, given D, the "knees" in the corresponding line plot help users identify good choices of k (e.g., $k = 9$ or 11 for $D = 1$), because additional clusters beyond these points would bring diminishing improvement to clustering quality. In contrast, the ranges of k within which the plot is close to linear or flat (e.g., $k > 6$ for $D = 3$) are less interesting.

3. Visualizing the evolution of clusters as parameters change. When users make some change to the clustering parameters k, L, and D, QAGView provides an option to visually compare the old and new result clusters, such as Figure 5. Recall that if an old cluster (rectangle on the left) and a new cluster (rectangle on the right) overlap in their contents, there will be a band connecting them. When the pointer hovers over an old cluster, for example in Figure 5, QAGView will highlight this cluster as well as the bands (three in this case) that connect to it, letting users easily see where its contents get regrouped under the new clustering. QAGView will also show details about the highlighted cluster, such as the number of aggregate result tuples it covers (size), the number of top-L result tuples it covers (coverage), and the average/maximum/minimum values of the covered tuples. QAGView shows details about each band connected to the highlighted cluster as well, such as the number of result tuples shared by the source and destination clusters, and the percentage of the result tuples in the source cluster that get regrouped to the destination cluster (note that the sum of percentages over all bands connected to the highlighted cluster may exceed 100%, as destination clusters may overlap).



Database List

- template1
- template0
- postgres
- movielens
- congress
- beers
- VoteCorr
- tpcds

Table List

- occupations
- users
- ratings
- movies
- genres
- genres_movies
- movies_complete
- big_rating_table
- r
- m_view

Table Preview

Choose a database, preview table schema and contents here!

movie_id	movie_title	release_date	video_release	imdb_url	genres_unknown	genres_action	genres_adventure	genres_animation	genres_children	genres_comedy	genres_crime	genres_documenta
1	Toy Story (1995)	1995-01-01	null	http://us.imdb.com/M/title-exact?Toy%20Story%20(1995)	0	0	0	1	1	1	0	0
1	Toy Story (1995)	1995-01-01	null	http://us.imdb.com/M/title-exact?Toy%20Story%20(1995)	0	0	0	1	1	1	0	0
1	Toy Story (1995)	1995-01-01	null	http://us.imdb.com/M/title-exact?Toy%20Story%20(1995)	0	0	0	1	1	1	0	0
1	Toy Story (1995)	1995-01-01	null	http://us.imdb.com/M/title-exact?Toy%20Story%20(1995)	0	0	0	1	1	1	0	0
1	Toy Story (1995)	1995-01-01	null	http://us.imdb.com/M/title-exact?	0	0	0	1	1	1	0	0

SQL Input

```
select extract(year from release_date) as year, extract(month from release_date) as month, age/10*10 as age_range, gender, occupation_name, genres_action, avg(rating) as val from big_rating_table group by year, month, age_range, gender
```

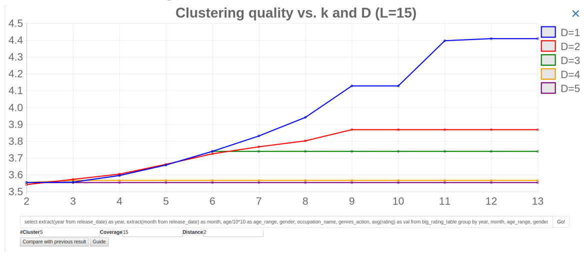
#Cluster 5 Coverage 10 Distance 2

SQL Result

After run SQL, Graph Shows here!

year	month	age_range	gender	occupation_name	genres_action	val	Expand
1995	8	20	M	Student	0	4.457142857142857	▼
*	1	*	M	Educator	0	3.8065282353001946	▼
*	1	20	M	Student	*	3.770025364490834	▼
*	1	20	M	Programmer	1	3.705734169250789	▼
*	1	*	M	Student	1	3.645958777788262	▼

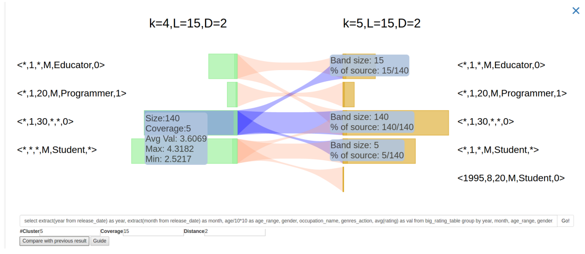
Figure 3: Main GUI of QAGView.



```
select extract(year from release_date) as year, extract(month from release_date) as month, age/10*10 as age_range, gender, occupation_name, genres_action, avg(rating) as val from big_rating_table group by year, month, age_range, gender
```

#Cluster 5 Coverage 15 Distance 2

Figure 4: Visualization of the effect of k and D on clustering quality (for a given L), to guide the selection of clustering parameters.



```
select extract(year from release_date) as year, extract(month from release_date) as month, age/10*10 as age_range, gender, occupation_name, genres_action, avg(rating) as val from big_rating_table group by year, month, age_range, gender
```

#Cluster 5 Coverage 15 Distance 2

Figure 5: Visualization of how clusters evolve as parameters change.

Acknowledgment. *This work was supported in part by NSF awards IIS-1408846, IIS-1552538, IIS-1703431, IIS-1718398, and NIH award 1R01EB025021-01.*

REFERENCES

[1] Marina Drosou and Evaggelia Pitoura. 2015. Multiple Radii DisC Diversity: Result Diversification Based on Dissimilarity and Coverage. *ACM Trans. Database Syst.* 40, 1, Article 4 (March 2015), 43 pages.

[2] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.* 5, 4, Article 19 (Dec. 2015), 19 pages.

[3] Manas Joglekar, Hector Garcia-Molina, and Aditya G. Parameswaran. 2016. Interactive Data Exploration with Smart Drill-Down. In *Proc. of the 2106 IEEE Intl. Conf. Data Engineering*. Helsinki, Finland, 906–917.

[4] Lu Qin, Jeffrey Xu Yu, and Lijun Chang. 2012. Diversifying Top-K Results. *Proc. of the VLDB Endowment* 5, 11 (2012), 1124–1135.

[5] Yuhao Wen, Xiaodan Zhu, Sudeepa Roy, and Jun Yang. 2018. Interactive Summarization and Exploration of Top Aggregate Query Answers. (2018). Manuscript, https://users.cs.duke.edu/~sudeepa/SummarizingAggregates.pdf.