TopKube: A Rank-Aware Data Cube for Real-Time Exploration of Spatiotemporal Datasets

Fabio Miranda, Lauro Lins, James T. Klosowski, and Claudio Silva

Abstract— From economics to sports to entertainment and social media, ranking objects according to some notion of importance is a fundamental tool we humans use all the time to better understand our world. With the ever-increasing amount of user-generated content found online, "what's trending" is now a commonplace phrase that tries to capture the zeitgeist of the world by ranking the most popular microblogging hashtags. However, before we can understand what these rankings tell us, we need to be able to more easily create and explore them, given the significant scale of today's data. In this paper, we describe the computational challenges in building a real-time visual exploratory tool for finding top-ranked objects, build on the recent work involving in-memory and rank-aware data cubes to propse TopKube, and demonstrate the usefulness of our methods using real-world, publicly available datasets.

1 Introduction

Ranks and lists play a major role in human society. It is natural for us to rank everything, from movies to appliances to sports teams to countries' GDPs. It helps us understand a world that is increasingly more complex by only focusing on a subset of objects. There is probably no better way to describe a decade than by ranking its most popular songs or movies. One just needs to look at the *Billboard Hot 100* of each year to grasp of how the majority of society used to think and behave.

With the ever-increasing amount of user-generated content found online, ranks have never been so popular to our cultural landscape. *"What's trending"* has become a commonplace phrase used to capture the spirit of a time by looking at the most popular microblogging hashtags. The same way that the most popular songs can be used to describe the *zeitgeist* of a year or a decade, *what's trending* can be used to describe the spirit of a day or even an hour.

The ubiquity of GPS-enabled devices provides further insight into the people creating this content by providing their location information, which creates a much more interesting, and more complicated, version of the ranking problem. Now, not only are we interested in what is trending over time, but also over space, which can range from the entire world all the way down to a city block. Creating techniques that compute these rankings and that allow for exploration of such ranks are thus increasingly important to better understand our world.

A system that can efficiently process a large number of records and come up with answers in a few minutes or seconds can be applied to an innumerable set of important problems, including those described here. In recent years, though, with the explosion of data gathering capabilities, there is a growing demand for a kind of tool for which latency at the scale of seconds or minutes is unacceptable. Problems for which no automatic procedure exists that can replace human investigation of multiple (visual) data patterns require exploratory tools that are most effective when driven by a low latency query solving engine. Making a human wait for the result and breaking her flow of thought might be the difference between capturing an idea or not.

We have recently seen a growth in research into techniques enabling low latency queries for the purpose of driving visual interactive interfaces. For example, from the perspective of enabling fast scanning of the data at query time, systems like MapD [11] and imMens [10] use the parallel processing power of GPUs to answer queries interactively. From a complementary perspective, Nanocubes [8] describe how to pre-aggregate data in an efficient but memory intensive fashion to allow for light-weight processing of queries also interactively.

In this paper we also follow this path of describing techniques for low-latency querying of large data for exploratory visualization purposes. We propose an extension to the efficient pre-aggregation scheme described in Nanocubes [8] to cover an important use case that was largely ignored and very inefficient in the original proposal: namely, interactively ranking the top-k objects from a large data collection using an arbitrary multi-dimensional selection. For example, we would like to answer queries such as: "What are the top-20 Flickr tags?" for a manually selected spatial region such as Europe, California, Chicago, or even Central Park. Furthermore, we want to be able to determine how the popularity of these tags evolved over time, as dictated by the end-user's interests, all at interactive rates. We show that the state-of-the-art is not able to compute such queries fast enough to allow for interactive exploration, but our method, called TopKube, is, up to an order of magnitude faster than the previous techniques.

Fabio Miranda and Claudio Silva are with NYU. Lauro Lins and James Klosowki are with AT&T Labs. E-mails: {fmiranda, csilva}@nyu.edu, {llins, jklosow}@research.att.com

2 Related Work

The challenge of visualizing large datasets has been extensively studied. Most techniques perform data reduction by aggregating a large number of points into as few points as possible, and then visualizing that smaller aggregation. These reductions, such as sampling [4], filtering [15], and binned aggregation [2], try to convey most of the properties of the original dataset while still being suitable for interactive visualization. Even though sampling and filtering reduce the number of items, certain aspects of the data (outliers) may be missed. As pointed out by Rousseeuw and Leroy [13], data outliers are an important aspect of any data analysis tool. Binned aggregation, however, does not have such limitations. The spatial domain will be divided into bins, and each data point will be aggregated into one of those bins. The visualization will be reduced to the number of bins in the domain.

The *visual* exploration of large datasets, however, adds another layer of complexity to the visualization problem. Now, one needs to query the dataset based on a set of user inputs, and provide a visual response as quickly as possible, in order not to impact the outcome of the visual exploration. In Liu and Heer [9], the authors present general evidence for the importance of low latency visualizations, citing that even a half second delay in response time can significantly impact observation, generalization, and hypothesis rates. Data structures such as imMens [10] and Nanocubes [8] leveraged a data cube to reduce the latency between user input and visualization response. Data cubes have been explored in the database community for a long time [6], but in the visualization community, they were only first introduced in 2002 by Stolte et al. [16, 17]. All of these techniques, however, are limited to simple data types, such as *counts*. They were designed to answer queries such as: *"How many pictures were uploaded from Paris during New Year's Eve?"*. Our data structure goes beyond that. We are able to answer more detailed queries, such as *"What were the most popular image tags for all the pictures uploaded from Paris?"*.

The notions of ranking and top-k queries were also first introduced by the database community. Chen et al. [3] presents a survey of the state-of-the-art in spatial keyword querying.The previous data

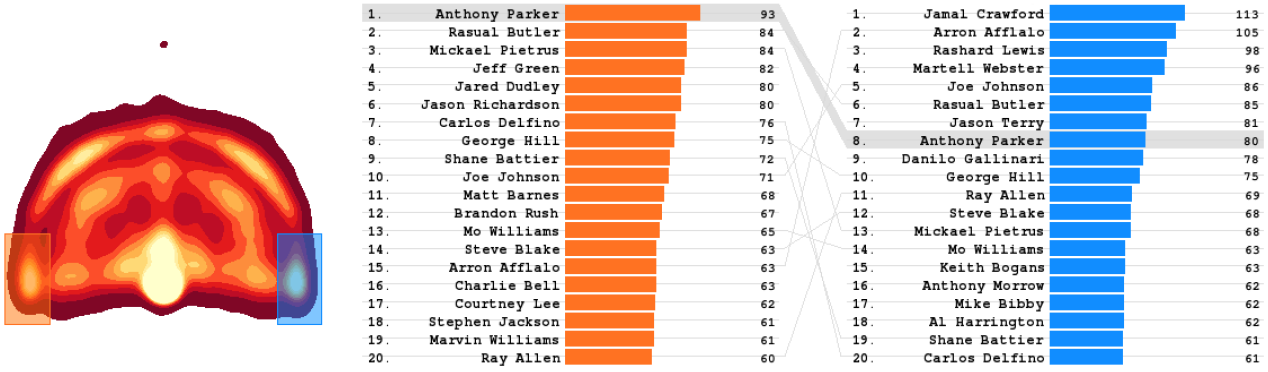

Fig. 1. Ranking NBA players by number of shots from the left 3-point corner (orange) and right 3-point corner (blue) for the 2009-2010 season. The left image is a heatmap of all shots: brighter colors indicate more shots were taken from that location. The hotspot clearly identifies the basket.

structures, however, focus on building indexing schemes suitable for queries where the universe of keywords is restricted. In other words, given a set of keywords, rank them according to their popularity in a region. If there is no keyword restriction (or the number of restricted keywords is too large), then such proposals become unfeasible. Our proposal is much broader: we are able to compute the rank of most popular keywords even if there is no keyword restriction.

Rank-aware data cubes have also been proposed in the database community. Xin et al. [19] proposed the *ranking cube*, a rank-aware data cube for the computation of top-k queries. Wu et al. [18] then proposed the ARCube, also a rank-aware data cube, but that supports partial materialization. Our proposal differs from them in two major ways: we specialize the data structure to better suit spatiotemporal datasets, and we demonstrate how our structure can provide low latency, real-time visual exploration of large datasets.

Another related database research area is top-k query processing: given a set of lists, where each element is a tuple with key and value, compute the top-k aggregated values. Several memory access scenarios led to the creation of a number of algorithms [7]. The NRA (no random access) algorithm [5] assumes that all lists are sorted and that the only memory access method is through sorted access. The Threshold Algorithm (TA) [12, 1] considers random access to calculate the top-k. More recently, Shmueli-Scheuer [14] presented a budget-aware query processing algorithm that assumed the number of memory reads is limited. We propose a different top-k query processing algorithm, suitable for our low latency scenario. We show that, due to the high sparsity of the merged ranks, past proposals are not suitable.

3 Binning and Counting

To motivate the discussion, we begin with a simple example. Assume a data analyst is studying shots in National Basketball Association (NBA) games from a table like the one below (only three rows shown).

team	player	time	pts	x	y
CLE	LeBron James	5	0	13	28
BOS	Rajon Rondo	5	2	38	26
CLE	LeBron James	7	3	42	35

In this table, every row represents a shot in a game. The first row, for instance, indicates that LeBron James, who plays for Cleveland, took a shot in the 5th minute of a game from the court coordinates $x = 13$, and $y = 28$; the shot missed the basket and he scored 0 points (`pts`). The second row represents a 2-point shot made by Rajon Rondo from Boston, and the third row represents the next shot taken in the game: a 3-point shot made by Lebron James. With a powerful querying interface (e.g. SQL) on top the table above, it is clear that an analyst can formulate many insightful aggregations. The query:

```
select player,count(player) as numshots from
table group by player order by numshots DESC
limit 50
```

would generate a ranked list of the players that took the most shots, while the following query would rank the players by points made.

```
select player,sum(pts) as numpts from table
group by player order by numpts DESC limit 50
```

The challenging task though is to be able to solve such queries at interactive rates for large datasets. If data is not indexed properly, queries might need to "touch" essentially all records in the database and result in queries consuming too much time for a proper interactive experience. To address this challenge there are basically two alternatives: (1) expand the computational power of the query engine by adding more processors scanning the data in parallel at query time, e.g. GPU computing; or (2) smart indexing of the data such that queries can be solved without a full or expensive scan of the data.

In order to speed up queries, the *Nanocubes* [8] authors follow approach (2), and observe that the careful encoding of aggregations in a sparse, pointer-based data structure is a fruitful alternative. They report on multiple real-life use cases where data cube materializations of large data sets could fit into the main memory of commodity laptops, enabling aggregation queries to be computed at interactive rates.

From a high level perspective, the Nanocubes approach is all about *binning* and *counting*. Each dimension of the data cube is modeled as a *bin hierarchy* (e.g. a spatial quad-tree, a categorical flat-tree). Given a table of records and a pre-defined binning scheme (i.e. bin hierarchies for each dimension), we can think that a combination of bins from different dimensions, or *product bins*, induce a set of records from the orignal table. In our example, if we think each team is a bin in the `team` dimension, and each player is a bin in the `player` dimension, the *product bin* `(CLE, L. James)` induces records 1 and 3 of the table. The nanocubes idea, as well as any explicit encoding of a data cube, is to pre-compute a certain measure of interest for each *product bin* based on the records that are incident to it. Here, the measure could be simply the number of shots `(CLE, L. James)` $\mapsto 2$, or the number of points `(CLE, L. James)` $\mapsto 3$.

The main drawback of an explicit data cubes as proposed by [8] is clear: even a minimal representation of a data cube tends to grow exponentially with the addition of new dimensions. On the other hand, from a practical perspective, there seems to exist a sweet spot in terms of computing resource utilization where the application of in-memory materialized data cubes can really be the driving engine of interactive exploratory experiences that would otherwise require prohibitively larger amounts of infrastructure.

4 TopKube

In this work we investigate the use of in-memory data cubes to drive an important use case for visualization that, to the best of our knowledge, was not fine tuned before. Namely, if we perform a multi-dimensional selection that results in millions of objects, how can we efficiently obtain a *list* of the top-k most relevant objects with respect to our measure of interest. For example, consider the selections we made on the

basketball court example in Figure 1. We want to compare the top-20 players that take shots from the left 3-point corner (orange) versus players that take shots from the right 3-point corner (blue). In this case, knowing that there are only a few hundred players in the NBA each year, it would not be computationally expensive to scan all players to figure the top 20, but there are many other cases such as GitHub projects, Flickr images, or microblog hashtags, where having to scan millions of objects can result in unacceptable latencies.

4.1 TopKube vs. Nanocubes

The use case considered by the original Nanocube data structure was that of multi-dimensional selections that resulted in a large number of data records, whose aggregated counts would be presented to the user in a variety of means: as pixel values on a heatmap, as categorical values in a barchart, or as temporal values in a time series line plot. The use case we have in mind here is different: the multi-dimensional selection in our case might result in hundreds of thousands to millions of object-value pairs, and we are not interested in presenting all these pairs to the user, but only the top valued pairs. More concretely, the problem we are interested in here is to quickly produce visalizations like Figure 1 even if the NBA had millions of players.

Each dimension in the original Nanocube is modeled as a *hierarchy of bins*, with the exception of *time*. Each *product bin*, i.e. the combination of one selection from each dimension, is instead mapped to a time series, which is implemented as a summed-area table. In TopKube, in order to speed up top-k queries, we propose that each *product bin* should be mapped not to a time series, but to a *rank-aware multi-set*. More formally, if β is a product bin, the original Nanocube would store a mapping like:

$$\beta \mapsto ((t_1, v_1), (t_2, v_1 + v_2), \ldots, (t_m, v_1 + \ldots + v_m)) \quad \text{[NANOCUBE]}$$

where t_i would be increasing time bins and v_i would be the measure of interest (e.g.record count). The cumulative values were stored there to allow for fast retrieval of value sums for any time interval. In the case of TopKube, we want each product bin β to be mapped to:

$$\beta \mapsto \left\{ \text{lst} = ((q_1, v_1, \sigma_1), \ldots, (q_j, v_j, \sigma_j)), \text{sum} = \sum_{i=1}^{j} v_i \right\} \quad \text{[TopKube]}$$

With this encoding, to access the value of a query key q in β we perform a binary search in *lst* (assuming it is ordered by q_i); the i-th top ranked object in β is the σ_i-th entry in *lst* and takes constant time (fast random access to σ_i + fast random access to k_{σ_i} and v_{σ_i}).

4.2 Top-K From Ranked Lists

With TopKube, we can easily produce a list of top-k ranked objects when a multi-dimensional selection results in a single product-bin β, but in general that does not happen. For example, in Figure 2, we show a common case in a spatiotemporal dataset: a 624 bin selection in space and 3 bins in time, which potentially results in a 1872 product bin selection. The pre-stored ranked lists we have for each β should help speed up the top-k query, but the task is not as trivial as collecting top-k resulting objects in $O(k)$ steps. To ease the exposition, and for the lack of a consistent name in the literature reviewed, we refer to this problem as *Top-k From Ranked Lists* or *TKFRL*.

4.3 Threshold Algorithm

The source of the difficulty for the TKFRL problem is that, for any key object q, its final measure v for our top-k ranking purposes might be broken into m summands $v = v_1, \ldots v_m$, one for each product-bin in the selection. Although we have an efficient way to access these summands in decreasing order (by putting all m lists into a heap/priority queue and popping the next largest key and summand), this does not directly imply we are going to find the measures for the top-k keys efficiently. Fortunately, a lot is known about the TKFRL problem [5]. The famous *threshold algorithm* or *TA* (which was explained and analyzed in the first database paper to win the prestigious Gödel Prize in 2014) is known to be optimal in a strong sense: no other algorithm

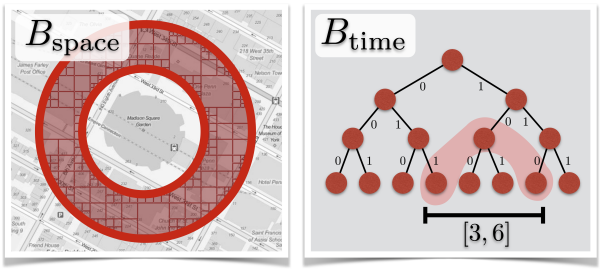

Fig. 2. Dimensions of *space* and *time* represented as bin hierarchies. B_{space} are bins in a quad-tree hierarchy: we show an annulus selection around Madison Square Garden corresponding to 624 bins; B_{time} is a binary hierarchy; we show 3 bins corresponding to the interval $[3, 6]$.

can access less data than the threshold algorithm does and still obtain the correct answer. The threshold algorithm consists of the following steps: (1) find key q of the next largest summand; after finding the other summands of q in the other $m-1$ β's, compute the key-value pair (q, v); (2) Insert the key-value pair found in the previous step into a buffer R that maintains only the top-k key-value pairs it has seen; (3) update threshold τ to be the sum of the available largest m summands (an upper bound for the total measure of a yet unseen key); (4) if R has k key-value pairs and the smallest valued pair is larger than τ, then report R as the top-k result, otherwise go to Step 1.

Although TA has ideal theoretical guarantees, there is an assumption that all m lists contain summands for all keys. This is natural given the application usually associated with TA: the m lists corresponded to m attribute-columns of a table and all keys (rows) should have an entry in each of those columns. However, the instances of the TKFRL problem that we observed were quite sparse: one key q is present in only a small fraction of the m lists, thus reducing the efficiency of TA.

4.4 Key Sweep Algorithm

Let us step back and suppose we do not store the ranking information, σ, in β. If we go back to a rank-unaware data structure, how can we solve the top-k problem? One way, which we refer to as as the *Naive Algorithm* is to traverse all the β's in the selection, and keep updating a dictionary structure of key-value pairs (we would increment the value of a key already in the dictionary with the current summand we found for that key in the current m-bin). Once we finish traversing all β's, we would sort the keys by their values and report the top-k ones. The Naive algorithm is correct, but inefficient. It uses memory proportional to all the keys in all m lists, and this number might be much larger than k (e.g. millions of keys instead of 100 if we ask for $k = 100$).

A more efficient way to do the union of m lists (that are sorted by keys) is to add all these lists into a heap/priority queue where the list with the smallest key is on the top of the heap. If we keep popping the next smallest key and summand from all the m lists, we will sweep all key-summand pairs in key increasing order, and every time we get a new (larger) key, we can be sure we know the total measure of all previous keys. Using this approach, we can maintain a result list with at most k buffers instead of a dictionary with all keys in all lists. We will refer to this approach as the *Key Sweep Algorithm*. Note that this algorithm scans all the summands, as does the Naive Algorithm, but it does not need a potentially large buffer to solve the top-k problem.

4.5 Hybrid Algorithm

The problem with the direct application of TA to solve the TKFRL problem is that in sparse instances, for each good candidate key to be in the top-k result, the algorithm performs a binary search for the other $m-1$ summands for that same key. If every key had a summand present in all m lists (*dense* instance), these cycles would be useful, but in a sparse instance of the problem, these are mostly wasted cycles. In typical instances of the TKFRL problem (e.g., what are the most active GitHub projects in the west coast of the U.S.?), we observe that on average each key is in less than 3% of the m-lists in the selection.

On the other hand, we note that the Key Sweep Algorithm wastes no cycles and only accesses summands that are present in the data. In order to get the best results in our experiments, we followed a hybrid approach between TA and the Key Sweep Algorithm.

If the problem instance is dense, our *Hybrid Algorithm* will simply run the TA directly. Otherwise, we use the Key Sweep Algorithm to merge the s smallest lists (from the m original lists) into one list s^* and then run TA with the largest $m-s$ lists plus s^*. The intuition is that we reduce the number of bins in order to raise the density of the TKFRL problem instance, and avoid all the access misses that pure TA would have. For example, think about the 624 spatial bins in Figure 2. We typically expect the smaller squares in that spatial selection to have less data than larger squares. The idea would be to merge all the lists of the smaller squares to make the problem instance dense for TA.

To detect the sparseness of a problem instance, we note that a lower bound for the number of keys is given by the length ℓ of the largest of the m lists. The total number of summands in a problem instance is the sum n of the lengths of all the m lists. Thus, an upper bound of the density of the TKFRL instance is given by $n/(m \times \ell)$. The idea of the hybrid approach is to replace m by a smaller m' to raise the density estimate to a certain desired level θ. We have experimented with different θs and get very good results with $\theta = 0.25$.

5 Experimental Results

Our system was implemented using a distributed client-server architecture. All rendering is performed on the client side, while the server is optimized to handle large-scale data efficiently and serve data requests to the clients. This design enables our system to scale well with an increase in the number of clients and/or data volume.

The server instance is powered by a C++11 executable that can receive and fulfill data requests over HTTP. It includes an API for client-server communication, thus, virtually supporting all types of clients, ranging from desktop to mobile applications. In our prototype, we implemented a browser-based client using HTML5, D3, and WebGL for its portability and the highly parallel architecture of GPUs.

5.1 Datasets

The datasets used in our case studies are freely available and allow the extraction of geotagged keywords: articles on Wikipedia, tags on Flickr, projects on Github, and hashtags on microblogs. Those keywords were then used as objects in the construction of our TopKube data structure. A summary of all datasets can be seen in Table 1: the number of objects, the amount of memory to build TopKube, the original size of the data file, the number of keywords, and the construction times are shown respectively.

Dataset	Objects	Mem	Size	Keys	Time
Wikipedia	112M	114G	1.2G	3.0M	5.2h
Flickr	84M	20G	532M	1.6M	3.9h
Github	58M	14G	329M	1.5M	3.2h
Hashtags	40M	53G	766M	4.7M	1.7h

Table 1. Summary of datasets used for our experiments.

5.2 Performance

In order to demonstrate the potential of TopKube and which algorithm works best when solving top-k queries in the interactive setups envisioned, we focus here on the Hashtags dataset, although performance across all datasets was similar. The Hashtags dataset has largest number of distinct keywords (4.7M) and the top-k problem instances coming from exploratory sessions on this dataset should tend to be harder (i.e. product-bins with longer lists of objects). As a baseline reference to estimate the TopKube performance, we also loaded the Hashtags dataset into a PostGIS table and created a mechanism to run the exact same queries in both engines (TopKube and PostGIS). The algorithms we implemented into TopKube were the Threhold Algorithm or TA (Section 4.3), Key Sweep Algorithm (Section 4.4), and the Hybrid Algorithm or (Section 4.5).

The experiment we ran was to collect 100 top-k queries from a interactive session where multiple spatiotemporal selections were chosen. These selections included large regions of space and time in order to stress the engine with harder problems. The companion video of this paper shows some of the queries we collected for this experiment. Once these queries were collected, we re-executed them in a serial fashion using the different algorithms inside TopKube and the PostGIS engine. The query results in all of these cases were the same. The total time to execute these queries in a serial fashion were: PostGIS $\mapsto$ 1h05min; TA $\mapsto$ 12min; Sweep $\mapsto$ 31.6s; Hybrid-0.75 $\mapsto$ 22.2s; Hybrid-0.50 $\mapsto$ 15sec; Hybrid-0.25 $\mapsto$ 8.3sec. It is clear from these results that neither PostGIS nor TA were fast enough to provide an interactive experience, while both Sweep and the different Hybrid runs were. In Figure 3 we show sorted query times for all the

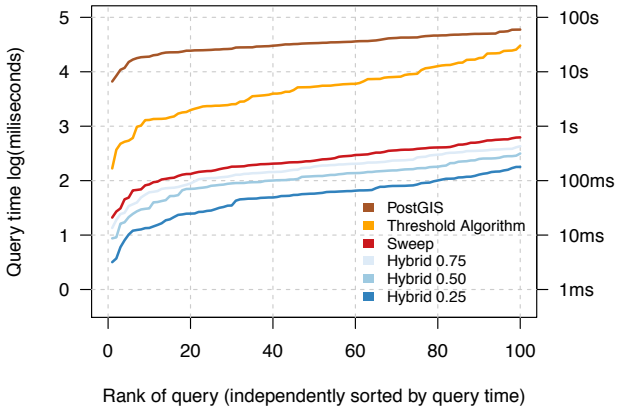

Fig. 3. Query times for top-k algorithms when solving 100 queries from an interactive session for $k = 32$ (see video). The x-coordinate i indicates the i-th fastest query for the corresponding algorithm.

100 queries from our experiment for each algorithm. With this plot we are able to see that the Hybrid algorithm with varying θ thresholds had query times consistently smaller then both TA and Sweep. This fact confirmed our hypothesis that top-k queries can be solved exactly in important use cases by adding rank information to the index. At the same time this fact seems obvious, this study shows that a natural use of rank information as done by TA does not yield a speed up. Only a combination of the strenghts of TA and Sweep illustrated by the Hybrid approach gave the speed up we expected. Although the lines in Figure 3 do not cross, a few of the queries were significantly slower in Hybrid than in Sweep. We plan to study these instances and find a stronger hybrid approach than the simple threshold we suggest here.

6 Conclusion

As user-generated online data continues to grow at incredible rates, ranking objects and information has never played such an important role in understanding our culture and the world. Although previous techniques have been able to create such rankings, they are inefficient and unable to be used effectively during an interactive exploration of the ranked data. We have introduced TopKube, an enhanced in-memory data cube that is able to generate ranked lists up to an order of magnitude faster than previous techniques. We have explored previous merging algorithms for creating these rankings, as well as designed improved algorithms for even greater interactivity. A careful experimentation of our techniques with multiple datasets has demonstrated its value. To date, we have not optimized the increased memory consumption of TopKube versus other recent in-memory data cubes. We believe there are opportunities to perform some compression of the internal data structure which could lead to significant memory savings.

REFERENCES

[1] W.-T. Balke and W. Kießling. Optimizing multi-feature queries for image databases. In *Proc. of the Intern. Conf. on Very Large Databases*, 2000.

[2] D. B. Carr, R. J. Littlefield, W. Nicholson, and J. Littlefield. Scatterplot matrix techniques for large n. *Journal of the American Statistical Association*, 82(398):424–436, 1987.

[3] L. Chen, G. Cong, C. S. Jensen, and D. Wu. Spatial keyword query processing: an experimental evaluation. *Proceedings of the VLDB Endowment*, 6(3):217–228, 2013.

[4] A. Dix and G. Ellis. by chance enhancing interaction with large data sets through statistical sampling. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, pages 167–176. ACM, 2002.

[5] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences*, 66(4):614–656, 2003.

[6] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Mining and Knowledge Discovery*, 1(1):29–53, 1997.

[7] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys (CSUR)*, 40(4):11, 2008.

[8] L. Lins, J. T. Klosowski, and C. Scheidegger. Nanocubes for real-time exploration of spatiotemporal datasets. *Visualization and Computer Graphics, IEEE Transactions on*, 19(12):2456–2465, 2013.

[9] Z. Liu and J. Heer. The effects of interactive latency on exploratory visual analysis. *Visualization and Computer Graphics, IEEE Transactions on*, 20(12):2122–2131, Dec 2014.

[10] Z. Liu, B. Jiang, and J. Heer. immens: Real-time visual querying of big data. In *Computer Graphics Forum*, volume 32, pages 421–430. Wiley Online Library, 2013.

[11] T. Mostak. An overview of mapd (massively parallel database). In *White paper*. Massachusetts Institute of Technology, 2013.

[12] S. Nepal and M. Ramakrishna. Query processing issues in image (multimedia) databases. In *Data Engineering, 1999. Proceedings., 15th International Conference on*, pages 22–29. IEEE, 1999.

[13] P. J. Rousseeuw and A. M. Leroy. *Robust regression and outlier detection*, volume 589. Wiley. com, 2005.

[14] M. Shmueli-Scheuer, C. Li, Y. Mass, H. Roitman, R. Schenkel, and G. Weikum. Best-effort top-k query processing under budgetary constraints. In *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*, pages 928–939. IEEE, 2009.

[15] B. Shneiderman. Dynamic queries for visual information seeking. *Software, IEEE*, 11(6):70–77, 1994.

[16] C. Stolte, D. Tang, and P. Hanrahan. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *Visualization and Computer Graphics, IEEE Transactions on*, 8(1):52–65, 2002.

[17] C. Stolte, D. Tang, and P. Hanrahan. Multiscale visualization using data cubes. *Visualization and Computer Graphics, IEEE Transactions on*, 9(2):176–187, 2003.

[18] T. Wu, D. Xin, and J. Han. Arcube: supporting ranking aggregate queries in partially materialized data cubes. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 79–92. ACM, 2008.

[19] D. Xin, J. Han, H. Cheng, and X. Li. Answering top-k queries with multi-dimensional selections: The ranking cube approach. In *Proceedings of the 32nd international conference on Very large data bases*, pages 463–474. VLDB Endowment, 2006.