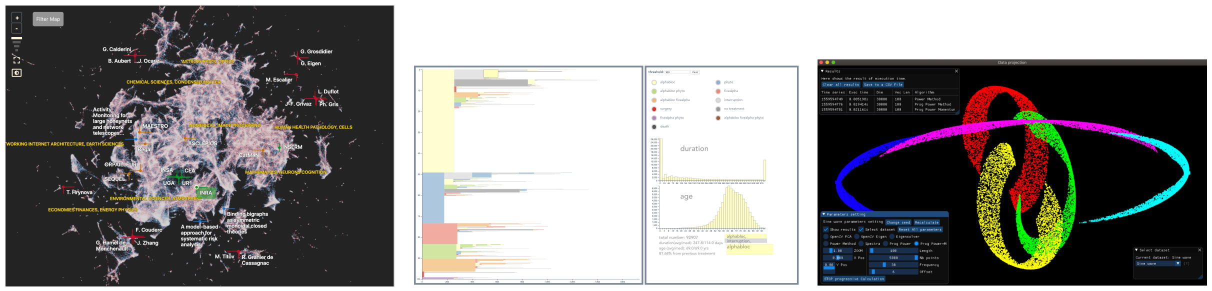
Practical Use Cases for Progressive Visual Analytics

Jean-Daniel Fekete[*]
Inria, France

Qing Chen[†]
École Polytechnique & Inria, France

Yuheng Feng[‡]
Inria, France

Jonas Renault[§]
CNRS, France



(a) Cartolabe: A Visualization System for Large Textual Corpora (b) ParcoursVis: EventFlow-style Visualization of Millions of Patient Records (c) PPCA: Progressive PCA for Massive Time-Series

Figure 1: Three Examples of Progressive Visual Analytics Systems

Abstract

Progressive Visual Analytics (PVA) is meant to allow visual analytics application to scale to large amounts of data while remaining interactive and steerable. The visualization community might believe that building progressive systems is difficult since there is no general purpose toolkit yet to build PVA applications, but it turns out that many existing libraries and data structures can be used effectively to help building PVA applications. They are just not well known by the visual analytics community. We report here on some of these techniques and libraries we use to handle "larger than RAM" data efficiently on three applications: Cartolabe, a system for visualizing large document corpora, ParcoursVis, a system for visualizing large event sequences from the French social security, and PPCA, a progressive PCA visualization system for large amounts of time-sequences. We explain how PVA can benefit from *compressed bitset* to manage sets internally and perform extremely fast Boolean operations, *data sketching* to compute approximate results over streaming data, and use *Online* algorithms to perform analyzes on large data.

1 Introduction

This paper presents three progressive visual analytics applications that allow visualization and interaction with large amounts of data. It explains how scalability can be achieved using software components that are relatively new and not well known by the visualization community, but can greatly help them building progressive systems.

The whole data science pipeline is undertaking a complete change; the tradition stack of layers used by environments such as R or Python is being challenged by new architectures that introduce novel capabilities to manage "out of core" data. We describe a few of these mechanisms here and explain how they are used in our scalable visualization systems. In particular, we discuss the following structures:

- *Compressed bitset* to manage sets internally, and perform extremely fast Boolean operations.
- *Data sketching* to compute approximate results over streaming data
- *Online algorithms* to perform computation by batches.

By using all these novel low-level components, progressive applications can be built effectively with a minimum of low-level development. Developing low-level mechanisms and algorithms for large-scale data is notoriously difficult to do well. It needs to be robust to resist to all the cases that occur in large data and it also need to be efficient to process the data as quickly as possible. These two properties are often perceived as challenging by students or researchers who, understandably, are afraid of spending time addressing engineering issues. We invite them to engage in applying visualization to larger datasets by relying on these components.

2 Progressive Data Analytics and Visualization

In 2014, Stolper et al. [29] proposed a general framework and workflow for Progressive Visual Analytics. They define it as: "statistical algorithms that produce semantically meaningful partial results that improve in quality over time." Since then, there has been several lines of research, related to better understand the requirements of PVA algorithms and applications, the development of PVA algorithms [13,22,23,26,28], their use in applications [11,21], studies on the ability of analysts to cope with progressive results [2,35], and implementation of progressive systems [7]. A recent Dagstuhl seminar was dedicated to "Progressive Data Analysis and Visualization" [8], producing a vision article on "Progressive Data Science" [32].

Yet, building a PVA application from scratch is difficult because the standard data science software stack supported by most languages and programming environments is not designed for progressive systems. The early PVA applications have indeed been implemented with little software support, requiring an intensive amount of coding. There will be multiple stages needed to facilitate the development of more PVA applications, from libraries to design patterns. But it is striking to realize that some components already exist and can be used as building blocks for PVA.

This article focuses on three of these building blocks: *compressed bitset*, *data sketching*, and *online algorithms*. They are useful pieces in the sense that they allow computing over very large amounts of data with limited memory and usually very efficiently. They can be used in multiple ways, not always envisioned by their authors, but useful for PVA.

*e-mail: jean-daniel.fekete@inria.fr
†e-mail: qing.chen@inria.fr
‡e-mail: yuheng.feng@inria.fr
§e-mail: jonas.renault@lri.fr

2.1 Compressed Bitset

A compressed bitset is a compressed array of bits, also called *bit vector* or *compressed bitmap* (a very misleading term in visualization that also uses graphical bitmaps). Compressed bitsets have become popular in the database literature, dating from Oracle "compressed bitmaps" [1, 34], to more recent attempts at improving speed and compression rates [15,16]. The idea of the most popular and efficient system to date, Roaring Bitmaps [16, 33], is to store a set of bits in a clever manner to allow fast Boolean operations and low memory consumption.

With such a data structure, it becomes very efficient to store data indexes as bitsets, and to perform Boolean operations for combining indexes between these bitsets. For example, the ElasticSearch [10] database translates a filter query into a bitset that can be cached for further uses. Combining filters is done by combining the cached bitsets, leading to large speedups compared to re-evaluating queries. For visualization, the results of dynamic queries such as range queries can be easily translated into compressed bitsets (e.g., the selected rows are $[1,5,6,10,100,\cdots]$) and updated efficiently. Furthermore, the bitsets can be saved in files and loaded quickly, so a set of pre-computed bitsets can be prepared in advance and reused for standard queries. Several other clever uses of compressed bitsets have been developed lately [20], and there is no doubt the visualization community will invent many new ones soon.

2.2 Data Sketching

Data sketching is a set of techniques used in data streaming operations to build a summary of the properties of the data that is retained when the whole data would be too large to store [5]. Most sketches are therefore used as data structure to query at any point and return approximate results with a bounded error. The first use of Sketches was to keep track of the number of unique values in a very long stream [9]. A very useful sketch for PVA keeps track of the quantiles of a stream of values, such as t-Digest [6] and KLL [14]. Stream quantile sketches are fed with a stream of values, and can be queried at any time to return the approximate value of any quantile, such as the median (0.5 quantile). Sketches can also be used to maintain the histogram of stream values. Many other sketches are ready to be used for PVA [5], thanks to the data streaming community.

2.3 Online Algorithms

An *Online Algorithm* can process its input piece by piece, in contrast to an *Offline algorithm* that needs all its input to perform its computation. They are very related to progressive algorithms, with a few differences regarding bounded-time guarantees that are required for PVA but not for online. Still, many online algorithms can be used for PVA by making sure their input is controlled so the computation time is also bounded. For example, Turkay [31] uses an online implementation of PCA to be able to compute PCA progressively on a large number of high-dimensional vectors. It is often possible to adapt an online algorithm to make it progressive with reasonable efforts [13], or none at all [31]. For example, the scikit-learn toolkit [19] provides 10 online machine-learning algorithms that can be used directly in a PVA application (Classification: Perceptron, SGD classifier, Naive bayes classifier; Regression: SGD Regressor, Perceptron, Passive Aggressive regressor, SAG Logistic regression; Clustering: Mini-batch k-means; Feature extraction: Mini-batch dictionary learning, Incremental PCA).

3 Application Scenarios

In this section, we propose three examples showing the usage scenarios for PVA, covering research publication data on the web, patient pathway from medical records data, and PCA visualization of the results of ensembles simulations producing time-series.

3.1 Cartolabe: A Progressive Visualization System for Large Textual Corpora

Cartolabe [4] is a web-based data visualization platform which allows users to explore a 2D map of textual documents. On publication data, users can search for authors, articles, lab, etc. to see where they fit on the map. To understand the underlying connections that shape a specific domain, users could also search for an item and look for the 20 nearest entities.

The Cartolabe project has been developed as a framework with two main components: a visualization component and a data processing component. The latter defines multiple NLP pipelines to transform text corpora into 2D maps (see Fig. 1a). The current pipelines include the French repository or research articles "HAL" (500k articles), Wikipedia (4.5m articles), the publications of the IEEE VIS conference "VisPubData" [12] (3k articles with abstract), and many more. The pipelines first extract the data from their original repository, apply NLP techniques to convert each document into a high-dimensional vector (100-1000 dimensions). The vectors are then projected to a 2D space using the UMAP dimension reduction technique [17]. UMAP is an effective algorithm to preserve short distances between similar documents and to maintain a global meaningful shape. Cartolabe also uses k-means clustering to generate regions and find meaningful labels from them. The regions are labeled in yellow in Fig. 1a. The output of the NLP component is a set of aligned tables: x, y coordinates computed from UMAP, and multiple columns containing extra information regarding the points at x, y such as their type, label, importance (a score), url, year, etc.

The visualization component takes the data computed by an NLP pipeline, prepares it for visualization, and serves web-based applications through a Python back-end. That is where our progressive mechanisms have been implemented.

During its preparation, the server pre-computes tiles from the x, y table, to allow tile-based rendering of the map. Up to this stage, all the operations are performed offline and can take a long time to execute without any impact on the user's experience. Therefore, in addition to the computation of a hierarchy of tiles, we also compute a set of bitsets using the RoaringBitmap library [16] to prepare popular selection queries. For the HAL dataset, we create a bitset per year that we save in a database. We also create a bitset for the main French institutions (e.g, CNRS, Inria, all the universities in France). Additionally, we create one bitset for indices of points lying in each tile to be able to recompute a filtered version of it on demand.

During the exploration using the web-based system, the user can ask to filter the data. We then create the results as tiles computed in a progressive way. The user can filter according to a mix of years and institutions. The server collects the query, computes the Boolean operations from the low-level bitsets (years, institutions) to build a result bitset from the query. The server then generates the tiles in a progressive way. Since we know the indices of the items that lie in each tile, we can intersect them with the query results, fetch their x, y values from the table and draw them on the new tiles. The tiles are created first within the viewport visible by the user, and then extended to allow panning. If the user zooms in or out, the desired tiles are created on demand, although in the future, we could try to predict the tiles to generate using techniques inspired from ForeCache [3]. Still, for the user, during pan and zoom navigation, the information is updated in a progressive manner at an acceptable speed, usually limited by the network connection and not by the computation speed of the tiles. Although some modern database techniques can be used to compute queries exactly and quickly [25], reconstructing graphical structures such as tiles take time and should be done on demand and progressively. Compressed bitsets offer the necessary speed and flexibility.

Our technique for progressive generation of tiles could also be used in tile-based visualization systems, such as ZVTM [24] or Kyrix [30], to enhance them to support dynamic queries.

3.2 Progressive Visual Analytics for Large-scale Patient Pathway from Medical Records Data

The second case applying progressive visual analytics is to analyze patients' treatment pathways and outcomes from electronic medical records data. In such a scenario, it is challenging for analysts to deal with the medical records data due to their sheer scale.

With a collaboration with the French social security (CNAMTS), we are allowed to extract and analyze data subsets from the nationwide database which is one of the largest health-care databases in the world, anonymously describing the whole history of medical reimbursement for 70 million beneficiary individuals. The analysts from CNAM want to analyze the patient pathways for specific treatments so as to understand how treatments are actually done in contrast to the recommendations. To achieve this goal, we are working on the design and the implementation of a visualization system, *ParcoursVis*, to explore its prescription data. We first transform the raw data into meaningful medical events and then visualize it interactively at scale. Fig. 1b shows a screenshot of our prototype with an aggregated view for the patient pathway and on-demand detailed information regarding the distribution of values (age, duration of the treatment, distribution of next events).

ParcoursVis is similar to EventFlow [18] but meant to scale to a much larger data size. Each patient is viewed as a sequence of medical events, such as, e.g., "drug-1, drug-2, surgery, end-of-treatment" for patient p1. Each event is related to one person, has a starting date, a ending date, a type, and additional details such as the patient's age, for a total of 4 mandatory columns, and as many additional columns as needed (age, other conditions, location, etc). When dealing with multiple patients, they are first aligned to a particular event, which can be the beginning of the treatment of interest, and then a prefix-tree is constructed. If the second patient (p2) events are "drug-1, drug-3, death", then the first event is the same as for p1 and a first node is created with the two patients with a distribution of ages (here age of p1 and p2), and treatment durations (duration of p1 and p2). Then two new nodes are constructed and linked to the first node, one related to drug-2, and the second to drug-3 etc. When all the patients are processed, the full tree is built. However, note that the event "end-of-treatment" is a synthetic event, in the sense that no drug prescription will create that event. It is created by looking at the patient's record and realizing that he has not taken any new drug related to the treatment of interest. Doctors are interested by treatment interruptions, that are also synthetic events related to a duration without treatment (e.g., 2 months). Therefore, building the event sequence requires rules that can be parameterized, and the user may want to explore the data through filtering, zooming, getting details on demand, and changing the parameters for constructing the event stream.

However, when analysts need to interact with the parameters controlling the generation of high-level medical events, or conduct search, or filtering, recomputing a new visualization is time-consuming as the aggregation and layout needs to traverse each data record. We cannot pre-compute all the aggregate views since some parameters require to be adjusted by the experts. Thus we need to recompute them on the fly, and this computation can take minutes to complete. Therefore, we are working on the progressive implementation of our event sequence visualization system. When you click on a node, which represents many events with the same prefix, you can see the histogram of ages and duration.

To achieve real-time interaction, we need to recompute the tree and the distribution in a progressive manner. To implement a progressive tree construction, we maintain two tables: one table for recording patient events, the other for recording global information such as age, gender, conditions, but also access the patient records in the first table. Then we can compute the tree progressively by constructing a prefix-tree with successive batches of patients drawn at random after shuffling the patient table. For the detailed distribution of values in each node, we delay its computation and provide it on-demand when the user clicks on a specific node. Before the end of the progressive computation, we cannot show the exact distribution of values but we can show an good approximation of it using a histogram sketch. We also need to compute the median duration for each node, since it is used to set its width. We use the t-Digest [6] sketch to maintain this information for each node, and return a quick approximation. Finally, to get the "details on demand" at each node, we keep a bitset of the patient indices.

3.3 Progressive PCA for Massive Time-Series

EDF, the French electricity provider, needs to forecast the behavior of complex hydraulic systems over time to take into account the evolution of the climate into its production and management of energy (e.g., managing dams and cooling nuclear plants). It uses a stochastic simulator with a parametric model taking p input parameters and generating n values for n time steps. EDF needs to study the behavior of the model around a target set of input parameters. For example, they may be interested by the evolution of the depth of a river over time. For that, the engineers generate multiple simulations by perturbing the parameters around the target values. This produces an ensemble of simulations (thousands to millions, see Fig. 1c) populated with members, i.e., one simulation with n time steps, to analyze the sensitivity of the simulation around the parameters. The question we want to address is related to grouping the results into clusters of values varying around a median curve. Ideally, all the resulting time-series (e.g. the depths) are clustered around one consistent behavior over time. However, it is also possible that multiple clusters of output results will be produced, corresponding to "modes" for the forecast.

To study these modes, the engineers gather M simulations and project them using a multidimensional projection (PCA to start with). They then cluster the time-series from the projections (Fig. 1c(b)) to get a group of curves that will be described through a median curve and a confidence interval. They can visualize each cluster using this median curve and CI as an envelope.

While this process is currently applied, it suffers from performance problems when the number of curves and the number of time-points increase. We want to allow interactive exploration of the parameter space from the envelopes: see which parameter set values produced the different clusters. The standard implementations of PCA are too slow for computing the results at a rate allowing data exploration. Our project investigates new progressive methods to compute PCA over large amounts of time-series in interactive time. For a start, we were able to use Incremental PCA [27], an online implementation of PCA that allows computing PCA with batches of time-series. By adjusting the batch size, we can control the latency of the progressive computation. We are working on improving the online algorithms by taking into account the continuity of values contained by the vectors when dealing with time-series. Yet, we were able to start the exploration from an existing online algorithm, as Turkay et al. [31] did a few years ago.

4 Conclusion

In this position paper, we have presented three practical use cases of progressive visual analytics. Progressive visual analytics has been developed for a few years but still lacks applications on real-world scenarios. We explained how some existing low-level structures and algorithms can be used today to greatly facilitate the implementation of PVA applications: compressed bitsets, data sketching, and online algorithms. We explained how these structures have been used in three progressive applications, saving us a lot of time.

There are still many challenges related to PVA that are not directly managed by these components, but it is important to keep in mind that developing PVA applications is possible immediately, often with reasonable development efforts.

References

[1] G. Antoshenkov. Byte-aligned bitmap compression. In *Proceedings of the Conference on Data Compression*, DCC '95, pp. 476–. IEEE Computer Society, Washington, DC, USA, 1995.

[2] S. K. Badam, N. Elmqvist, and J.-D. Fekete. Steering the craft: UI elements and visualizations for supporting progressive visual analytics. *Computer Graphics Forum*, 36(3):491–502, 2017.

[3] L. Battle, R. Chang, and M. Stonebraker. Dynamic Prefetching of Data Tiles for Interactive Visualization. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pp. 1363–1375. ACM, New York, NY, USA, 2016. doi: 10.1145/2882903.2882919

[4] The Cartolabe project. `https://cartolabe.fr/`. Last access on July 2019.

[5] G. Cormode. Data sketching. *Commun. ACM*, 60(9):48–55, Aug. 2017. doi: 10.1145/3080008

[6] T. Dunning and O. Ertl. Computing Extremely Accurate Quantiles Using t-Digests. *arXiv preprint arXiv:1902.04023*, 2019.

[7] J. Fekete and R. Primet. Progressive analytics: A computation paradigm for exploratory data analysis. *CoRR*, abs/1607.05162, 2016.

[8] J.-D. Fekete, D. Fisher, A. Nandi, and M. Sedlmair. Progressive Data Analysis and Visualization (Dagstuhl Seminar 18411). *Dagstuhl Reports*, 8(10):1–40, 2019. doi: 10.4230/DagRep.8.10.1

[9] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31(2):182 – 209, 1985. doi: 10.1016/0022-0000(85)90041-8

[10] A. Grand. Frame of Reference and Roaring Bitmaps, 2015. Last access on July 2019.

[11] T. Höllt, N. Pezzotti, V. van Unen, F. Koning, E. Eisemann, B. Lelieveldt, and A. Vilanova. Cytosplore: Interactive Immune Cell Phenotyping for Large Single-Cell Datasets. *Computer Graphics Forum*, 35(3):171–180, 2016. doi: 10.1111/cgf.12893

[12] P. Isenberg, F. Heimerl, S. Koch, T. Isenberg, P. Xu, C. Stolper, M. Sedlmair, J. Chen, T. Möller, and J. Stasko. vispubdata.org: A Metadata Collection about IEEE Visualization (VIS) Publications. *IEEE Trans. on Vis. and Comp. Graph.*, 23, 2017. To appear. doi: 10.1109/TVCG.2016.2615308

[13] J. Jo, J. Seo, and J.-D. Fekete. PANENE: A Progressive Algorithm for Indexing and Querying Approximate k-Nearest Neighbors. *IEEE Trans. on Vis. and Comp. Graph.*, pp. 1–1, 2018. doi: 10.1109/TVCG.2018.2869149

[14] Z. Karnin, K. Lang, and E. Liberty. Optimal Quantile Approximation in Streams. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 71–78, Oct 2016. doi: 10.1109/FOCS.2016.17

[15] A. Kuznetsov. The BitMagic C++ library. `https://github.com/tlk00/BitMagic`, 2019. Last access on July 2019.

[16] D. Lemire, O. Kaser, N. Kurz, L. Deri, C. O'Hara, F. Saint-Jacques, and G. S. Y. Kai. Roaring bitmaps: Implementation of an optimized software library. *Softw., Pract. Exper.*, 48(4):867–895, 2018. doi: 10.1002/spe.2560

[17] L. McInnes, J. Healy, and J. Melville. UMAP: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

[18] M. Monroe, R. Lan, H. Lee, C. Plaisant, and B. Shneiderman. Temporal event sequence simplification. *IEEE Trans. on Vis. and Comp. Graph.*, 19(12):2227–2236, 2013.

[19] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[20] N. Pezzotti, J.-D. Fekete, T. Höllt, B. Lelieveldt, E. Eisemann, and A. Vilanova. Multiscale Visualization and Exploration of Large Bipartite Graphs. *Computer Graphics Forum*, 37(3):12, June 2018.

[21] N. Pezzotti, T. Höllt, J. Van Gemert, B. P. Lelieveldt, E. Eisemann, and A. Vilanova. DeepEyes: Progressive Visual Analytics for Designing Deep Neural Networks. *IEEE Trans. on Vis. and Comp. Graph.*, 24(1):98–108, Jan 2018. doi: 10.1109/TVCG.2017.2744358

[22] N. Pezzotti, T. Höllt, B. Lelieveldt, E. Eisemann, and A. Vilanova. Hierarchical Stochastic Neighbor Embedding. *Computer Graphics Forum*, 35(3):21–30, 2016. doi: 10.1111/cgf.12878

[23] N. Pezzotti, B. P. Lelieveldt, L. van der Maaten, T. Höllt, E. Eisemann, and A. Vilanova. Approximated and user steerable tSNE for progressive visual analytics. *IEEE Trans. on Vis. and Comp. Graph.*, 23(7):1739–1752, 2017.

[24] E. Pietriga. A Toolkit for Addressing HCI Issues in Visual Language Environments. *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 00:145–152, 2005. doi: 10.1109/VLHCC.2005.11

[25] F. Psallidas and E. Wu. Smoke: Fine-grained Lineage at Interactive Speed. *PVLDB*, 11(6):719–732, 2018. doi: 10.14778/3184470.3184475

[26] V. Raveneau, J. Blanchard, and Y. Prié. Progressive sequential pattern mining: steerable visual exploration of patterns with PPMT. In *Visualization in Data Science (VDS at IEEE VIS 2018)*. IEEE, Berlin, Germany, Oct. 2018.

[27] D. A. Ross, J. Lim, R.-S. Lin, and M.-H. Yang. Incremental learning for robust visual tracking. *Int. J. Comput. Vision*, 77(1-3):125–141, May 2008. doi: 10.1007/s11263-007-0075-7

[28] S. Servan-Schreiber, M. Riondato, and E. Zgraggen. ProSecCo: Progressive Sequence Mining with Convergence Guarantees. In *Proceedings of the IEEE International Conference on Data Mining*. IEEE Computer Society, IEEE, Singapore, 2018.

[29] C. D. Stolper, A. Perer, and D. Gotz. Progressive Visual Analytics: User-Driven Visual Exploration of In-Progress Analytics. *IEEE Trans. on Vis. and Comp. Graph.*, 20(12):1653–1662, Dec 2014. doi: 10.1109/TVCG.2014.2346574

[30] W. Tao, X. Liu, Y. Wang, L. Battle, C. Demiralp, R. Chang, and M. Stonebraker. Kyrix: Interactive Pan/Zoom Visualizations at Scale. *Computer Graphics Forum*, 38(3):529–540, 2019. doi: 10.1111/cgf.13708

[31] C. Turkay, E. Kaya, S. Balcisoy, and H. Hauser. Designing progressive and interactive analytics processes for high-dimensional data analysis. *IEEE Trans. on Vis. and Comp. Graph.*, 23(1):131–140, 2017.

[32] C. Turkay, N. Pezzotti, C. Binnig, H. Strobelt, B. Hammer, D. A. Keim, J. Fekete, T. Palpanas, Y. Wang, and F. Rusu. Progressive Data Science: Potential and Challenges. *CoRR*, abs/1812.08032, 2018.

[33] J. Wang, C. Lin, Y. Papakonstantinou, and S. Swanson. An Experimental Study of Bitmap Compression vs. Inverted List Compression. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, pp. 993–1008, 2017. doi: 10.1145/3035918.3064007

[34] K. Wu, K. Stockinger, and A. Shoshani. Breaking the curse of cardinality on bitmap indexes. In *Proceedings of the 20th International Conference on Scientific and Statistical Database Management*, SSDBM '08, pp. 348–365. Springer-Verlag, Berlin, Heidelberg, 2008. doi: 10.1007/978-3-540-69497-7_23

[35] E. Zgraggen, A. Galakatos, A. Crotty, J.-D. Fekete, and T. Kraska. How progressive visualizations affect exploratory analysis. *IEEE Trans. on Vis. and Comp. Graph.*, 23(8):1977–1987, Aug 2017. doi: 10.1109/TVCG.2016.2607714