Reminiscences on Influential Papers

This issue's contributors cover papers that focus on different aspects of accessing *data*: RDFs, approximate nearest neighbor search, and compact hash tables. Furthermore, they all highlight the impact of the papers not only on their own research and career but also for the community in general. Enjoy reading!

While I will keep inviting members of the data management community, and neighboring communities, to contribute to this column, I also welcome unsolicited contributions. Please contact me if you are interested.

Pınar Tözün, *editor*
IT University of Copenhagen, Denmark
pito@itu.dk

Zoi Kaoudi
IT University of Copenhagen, Denmark
zoka@itu.dk

Thomas Neumann, Gerhard Weikum.
RDF-3X: a RISC-style engine for RDF.
In Proceedings of the VLDB Endowment, Volume 1, Issue 1, pages 647-659, 2008.

At the time that this paper [7] was published, I was doing my PhD and I was (feeling) part of the Semantic Web community. My topic was on distributed RDF query processing, optimization, and reasoning, so the paper was quite relevant to my work. Back then, there had been several proposals on different centralized RDF stores mostly from the Semantic Web community. Then, the RDF-3X paper appeared at VLDB and it really made a difference for me (and probably others). Its solution seemed very simple and elegant and its performance on runtime and scalability was remarkable. Although I continued my PhD by publishing in Semantic Web conferences, this paper, somehow subconsiously, played a role into deciding to change my career after my PhD and become part of the database community.

The paper proposed RDF-3X, an efficient and scalable RDF store engine. In contrast to most of the works so far that were proposing to store triples by splitting them in one table per property, it proposed to store the triples into a single gigantic 3-column table. Importantly, RDF-3X used all possible column permultations as indices together with dictionary encoding, and compression. This idea is so simple yet so powerful that made an impression on me. Later on, when I had already joined the database community, I also heard that the proposal of exhaustive indexing led to many interesting and controversial discussions among database researchers!

In the paper, the authors showed that having all these indices stored not only improves data access but makes the entire query processing faster. RDF-3X query processor followed a RISC-style design philosophy: it relied mostly on merge joins over sorted index lists. This was made possible thanks to its exhaustive indexing scheme. RDF-3X also encompassed a cost-based query optimizer to determine the right join ordering. It achieved very efficient cardinality estimation by utilizing the extensive indices and by maintaining an additional set of RDF-specific statistics for joins. The evaluation results showed huge performance benefits over simply loading the RDF data into a column-based or a row-based relational database. Interestingly, none of the open-source systems provided by the Semantic Web community could scale to the dataset sizes used in the experiments.

To conclude, I believe this paper has been very influencial across two different research communities (Semantic Web and databases) thanks to its simple idea and effective results. The fact that the system was also available to use gave the opportunity to

many researchers to come up with multiple follow-up works. I find it very inspiring to see simple ideas having such large impact in our research communities. I hope everyone takes that into account when reviewing papers nowadays.

Fatemeh Nargesian
University of Rochester, NY, USA
fnargesian@rochester.edu

Aristides Gionis, Piotr Indyk, Rajeev Motwani.
Similarity Search in High Dimensions via Hashing.
In Proceedings of the International Conference on Very Large Data Bases, pages 518-529, 1999.

When I saw Pınar's email asking for a couple of paragraphs for this column, one clear choice came to mind - and it remained unchanged despite all my procrastination. This paper [3] belongs to the line of work on Approximate Nearest Neighbor (ANN) Search.

The paper introduces the Locality-Sensitive Hashing (LSH) technique for approximate similarity search in high-dimensional spaces. The motivation is the inefficiency of traditional nearest neighbor search methods as dimensionality increases - a phenomenon known as the curse of dimensionality. The paper defines a family of hash functions for Euclidean distance such that the probability of collision is much higher for closer items than for others. These hash functions help build small-footprint signatures, consisting of hash values, that are similarity-preserving. This property enables the efficient retrieval of similar items without exhaustive comparisons. LSH achieves this by mapping fragments of signatures to buckets using standard hash functions. During query time, only the subset of buckets that have the same signatures as the query are searched. This drastically reduces computation cost. The paper provides provable guarantees on query time and approximation quality, specifically targeting (r, c·r)-nearest neighbor search (i.e., returning a point within distance c·r if any point exists within r).

I used various nearest neighbor search algorithms for my PhD research to overcome the scalability and efficiency challenges of data search in open repositories. While I always found these algorithms neat and useful, it was not until the thesis and paper-writing phase that I truly began to appreciate the elegance of LSH techniques. And, it was not until I started teaching the fundamentals of ANN search to my students that I came to appreciate the knitty gritty details of this paper and its previous and follow-up work around it.

Following the body of work that has built upon and around this paper has taught me a mindset and connected me to a broader world of practical research problems. While the VLDB'99 paper focuses on Euclidean space, the family of LSH has been developed for a variety of similarity measures: Cosine, Dot Product, etc. In recent years, indexes for approximate nearest neighbor search have regained popularity due to vector databases. More recent ANN indexes are shown to be more efficient and scalable in practice. Graph-based techniques, such as HNSW [6], NSG [2], and DiskANN [5], achieve search times of (poly/)logarithmic complexity by building proximity graphs with long-range and short-range links. The inverted index-based techniques such as FAISS, build an inverted index on the centroid of data partitions and only search for nearest neighbors within the partition associated with the closest centroid. Some of these techniques are in-memory and some are disk-based; some use product quantization and compression to reduce memory footprint; and, almost all scale to benchmark datasets of billions of points and are deployed in industry applications. The main difference, however, between the LSH family and the new generation of ANN, as shown by recent studies [4], is in their worst-case performance. The VLDB'99 paper shows that LSH has truly sublinear search time dependence on data size. Whereas, almost all others, with the exception of DiskANN, suffer from worst-case linear search time. Even DiskANN, which offers an improved worst-case complexity, does so at the cost of very slow preprocessing.

In the era of transformers and super-fast search techniques and all that give us great results and spark our curiosity to ask "why does it work and when not?", this paper has remained, for me, an example to follow in my own research; a reminder for when settling on an approximation for time-tradeoff, think about the guarantees of how fast and approximate our approximation is.

Niv Dayan
University of Toronto, Canada
nivdayan@cs.toronto.edu

John G. Cleary.
Compact hash tables using bidirectional linear probing.

In IEEE Transactions on Computers, Volume C-33, Issue 9, pages 828-834, 1984.

My first encounter with Bloom filters was a love at first sight. A Bloom filter is a space-efficient probabilistic data structure that allows you to test whether a key is definitely not in a set, or possibly is. By compressing a set of keys into a compact bit array in memory, Bloom filters enable fast membership tests that help avoid expensive storage or network lookups when the key in question is absent.

Originally proposed in 1970, Bloom filters have become a mainstay in modern systems. Yet, despite their widespread use, they suffer from two important limitations. First, they do not support deletions or dynamic expansion as the dataset grows. Second, they only support point queries-checking for the presence of individual keys-but not range queries, which are essential in many database applications that need to determine whether an entire range is empty.

Over the past few years, our lab has been exploring ways to overcome these limitations. A key source of inspiration in our journey has been the paper "Compact Hash Tables Using Bidirectional Linear Probing" by John G. Cleary from 1984 [1]. This paper presents a compact hash table design built on four major ideas:

1. Quotienting – storing only the suffix of a key, inferring the prefix from its location.

2. Robin Hood Hashing – resolving hash collisions by searching sequentially for a nearby available slot, while keeping colliding entries adajcent.

3. Dual bitmaps – marking the start and end of clustered entries that map to the same slot.

4. Prefix sum arrays – aggregating the 1s in the bitmaps to support efficient navigation and search.

I appreciate this paper not only for its technical contributions, but also for its clarity—it explains complex concepts in intuitive, problem-driven terms without compromising on rigor. Many filter data structures developed over the past 15 years have drawn from this framework to provide more memory-efficient alternatives to Bloom filters that also support deletions.

These ideas have directly informed our research. In our InfiniFilter (SIGMOD 2023) and Aleph Filter (VLDB 2024) papers, we designed filters that can dynamically expand while maintaining a stable false positive rate. We achieve this by using variable-sized fingerprints padded with unary codes. Supporting this feature required a hash table whose collision resolution does not depend on the fingerprints themselves, unlike in Cuckoo filters. The Cleary data structure was an ideal fit.

Our recent work on range filters has also benefited from this foundation. Memento Filter (SIGMOD 2025), for instance, stores variable-length payloads alongside keys—something that's naturally supported by Robin Hood hashing, where entries can simply be pushed and pulled in sequence. Our newest range filter, Diva (currently under submission to VLDB 2025), goes a step further by encoding key infixes and relying on the Cleary structure being order-preserving. Interestingly, the Cleary data structure wasn't originally proposed with these use-cases in mind, yet it turns out to be exactly what we needed for each of the above projects.

We would also like to acknowledge the influential paper "A General-Purpose Counting Filter: Making Every Bit Count" by Pandey et al. from SIGMOD 2017 [8]. This work demonstrated how to search similar structures using rank and select primitives implemented efficiently using CPU instructions introduced in Intel's Haswell line of processors (Bit Manipulation Instruction Set 2). It also showed how to succinctly encode counters to allow representing multisets. We built on top of this excellent design and its codebase in many of our projects.

Of course, these are just a few highlights among the many foundational papers that have shaped our thinking. Our pursuit may be compact data structures, but the foundation beneath them is anything but small.

1. REFERENCES

[1] J.G. Clerry. Compact Hash Tables Using Bidirectional Linear Probing. *IEEE Transactions on Computers*, C-33(9):828–834, 1984.

[2] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. Fast approximate nearest neighbor search with the navigating spreading-out graph. *Proc. VLDB Endow.*, 12(5):461–474, 2019.

[3] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity Search in High Dimensions via Hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, page 518–529, 1999.

[4] Piotr Indyk and Haike Xu. Worst-case Performance of Popular Approximate Nearest Neighbor Search Implementations: Guarantees and Limitations. In *Proceedings of the 37th International Conference on Neural*

Information Processing Systems, NIPS '23, 2023.

[5] Ravishankar Krishnaswamy, Magdalen Dobson Manohar, and Harsha Vardhan Simhadri. The DiskANN library: Graph-Based Indices for Fast, Fresh and Filtered Vector Search. *IEEE Data Eng. Bull.*, 48(3):20–42, 2024.

[6] Yury A. Malkov and Dmitry A. Yashunin. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. *CoRR*, abs/1603.09320, 2016.

[7] Thomas Neumann and Gerhard Weikum. RDF-3X: a RISC-style engine for RDF. *Proc. VLDB Endow.*, 1(1):647–659, 2008.

[8] Prashant Pandey, Michael A. Bender, Rob Johnson, and Rob Patro. A General-Purpose Counting Filter: Making Every Bit Count. In *Proceedings of the 2017 ACM International Conference on Management of Data*, page 775–787, 2017.