Research Statement and Agenda

Prashant Pandey
prashant.prashn@gmail.com

My goal as a researcher is to advance the theory and practice of resource-efficient data structures and employ them to democratize complex and large-scale data analyses. I have worked on designing and building tools for large-scale data management problems across computational biology, stream processing, and storage. I have pursued three basic strategies for large-scale data management: shrink it, organize it, and distribute it. For applications that can afford a small, bounded fraction of errors in results, I lossily shrink data in favor of space and speed. For applications where accuracy is critical, I organize data based on the underlying storage to achieve better performance. Finally, for applications where data is processed across multiple machines, I partition and distribute data in a way to minimize inter-machine communication.

In data structures, my work shows how to overcome trade-offs among space, time, and features in traditional hash-based data structures (filters, count sketches, hash maps) and build feature-rich, high-performance, and scalable data structures. This work is described in three papers in top-tier conferences (**SIGMOD, ESA**) [13, 21, 23]. In graph processing, my work shows how to build a streaming graph processing system that enables faster graph computations than state-of-the-art static systems while supporting fast updates. This work is described in a paper at a top-tier conference (**SIGMOD**) [26].

In computational biology, my work is having a transformative impact, because I am essentially rebuilding key parts of genomic and transcriptomic analysis tool-chains around data structures of my design and gaining significant performance improvements over state-of-the-art tools. This work is described in ten papers in flagship computational biology conferences (**RECOMB, ISMB, WABI**) and journals (**Cell Systems, Genome Biology, Bioinformatics, JCB**) [1–4, 17–20, 22, 24]. In stream processing, my work shows how we can use external memory for problems that have traditionally been analyzed in the streaming setting, enabling solutions that can scale beyond the provable limits of fast RAM. It builds a bridge between the worlds of external-memory and streaming algorithms. This work is described in two papers in a top-tier conference and journal (**SIGMOD, TODS**) [6, 25]. In storage systems, my work shows how to use modern data structures to overcome decades-old trade-offs in file-system design, yielding orders-of-magnitude performance gains. This work is described in five papers in two top-tier conferences (**FAST, SPAA**) and two journals (**TOS, TOPS**) including a Best Paper Award at **FAST 2016** [7, 15, 16, 30, 31].

Filter and sketch data structures. Much of my work in computational biology, streaming, and storage builds on my research into space-efficient and fast data structures, such as, filters, count sketches, and hash maps.

Filters are approximate membership query (AMQ) data structures that maintain a probabilistic representation of a set or multiset, saving space by allowing queries occasionally to return a false-positive. AMQs such as the Bloom filter [8], are almost five decades old and have been a workhorse in numerous applications in databases, storage systems, networks, computational biology, and other domains. However, applications often work around limitations in the capabilities or performance of current AMQs, making these applications more complex and less performant. I developed the Counting Quotient Filter (CQF) [21] which supports approximate membership testing and, unlike Bloom filters, counting the occurrences of items in a dataset (even for datasets with a skewed distribution). The CQF offers order-of-magnitude faster operations than Bloom filters. Moreover, the CQF has features that other AMQs lack, e.g., deletions, resizing, and efficiently scaling out-of-RAM. I further extended the CQF to develop a compact hash map for efficiently indexing small key-value pairs. The CQF-based hash map has been the main workforce for building large-scale indexes for biological and cyber streaming data. Based on my research, several top teams across academia and industry have replaced Bloom filters in their code with counting quotient filters.

Count sketches maintain a probabilistic and lossy representation of a multiset and return the count of an item with a small, configurable overestimation error. The count-min sketch (CMS) [11] is the most commonly used count sketch and has been extensively used to answer heavy hitters, top-k, and other popularity measure queries, the central problems in the streaming context, where people are interested in extracting the essence from impractically large amount of data. However, the overestimate in the CMS grows proportionally with the number of items N making the overestimation error too high for large datasets. Moreover, the CMS has poor scalability out of RAM which makes it unusable on disk. To keep the estimation error sublinear in N (e.g., $1/\log N$ or $1/\sqrt{N}$) which is often required

by applications makes the CMS too large to fit in memory even for moderately large values of N. I developed the buffered count-min sketch [13], an SSD variant of the traditional count-min sketch, that efficiently scales to SSDs and enables efficiently ingesting and estimating large-scale data while keeping the overestimation error sublinear in N.

Graph data structures. Existing static graph-processing systems are optimized for graph skewness to achieve high performance and low space usage by preprocessing a cache-efficient graph partitioning based on vertex degree [9, 33, 34]. In the streaming setting, the whole graph is not available upfront, however, so finding an optimal partitioning is not feasible in the presence of updates. As a result, existing streaming graph-processing systems take a "one-size-fits-all" approach, leaving performance on the table. I developed Terrace [26], a system for streaming graphs that uses a hierarchical data structure design to store a vertex's neighbors in different data structures depending on the degree of the vertex. This multi-level structure enables Terrace to dynamically partition vertices based on their degrees and adapt to skewness in the underlying graph. Terrace supports faster batch insertions compared to Aspen [12], a state-of-the-art graph streaming system. On graph query algorithms, Terrace is faster than Aspen and offers similar performance as Ligra [28], a state-of-the-art static graph-processing system.

Computational biology. In computational biology, I showed how to scale genomic and transcriptomic indexes to petabyte-scale data using recent theoretical advances in compact and succinct data structures.

I first developed Squeakr [22] to solve the k-mer counting problem. k-mer counting involves counting the number of occurrences of each k-mer (a length-k substring) in a sequencing dataset. It is one of the most common preliminary step in many bioinformatics analyses. It is used to weed out erroneous data caused by sequencing errors, to estimate sequencing depth, to prepare sequencing data for assembly, and many other tasks. Squeakr uses the CQF [21], a counting data structure we developed, to count k-mers. Squeakr further extended the CQF to support thread-safe counting and offers efficient scalability with increasing number of threads even for highly skewed data. Squeakr achieved an order-of-magnitude faster queries and at the same time up to $4\times$ faster construction compared to the state-of-the-art k-mer counters, while also producing smaller indexes.

I then developed deBGR [20], which efficiently represents weighted de Bruijn graphs which are at the heart of many sequence analyses [10, 27]. In computational biology, the de Bruijn graph (dBG) is used to represent k-mer content in sequencing datasets as a weighted graph, where each k-mer represents an edge connecting its two $(k-1)$-mer substrings (nodes) and the k-mer count represents the edge weight. De Bruijn graphs are used in many biological analyses, however, they tend to require a substantial amount of memory for large data sets making many analyses slow or even impossible to run on a single machine. deBGR uses the CQF to store an approximate representation of the weighted de Bruijn graph in much smaller space compared to exact representations. It then uses an error-correction algorithm that exploits invariants of weighted de Bruijn graphs to iteratively correct all the approximation errors, making it practical to perform large-scale de Bruijn graph analyses in RAM.

I also developed a sequence-search index Mantis [1–3, 18, 19], which attempts to solve the fundamental problem of sequence-search over immense collections of raw sequencing datasets. The ability to perform sequence searches is so essential that BLAST [5], an earlier tool for sequence-search on assembled data, is one of the most cited papers in the scientific literature. But BLAST works only on assembled data, not raw data. And the process of converting raw data into assembled data is expensive, so the vast majority of raw data never gets assembled, making it inaccessible to BLAST-based searches. Mantis overcomes this limitation and enables quickly searching through thousands of raw sequencing experiments, constituting terabytes of data. The ability to perform searches on raw sequencing data would enable us to answer lots of questions that are not possible to answer by looking at assembled data [19, 29]. Prior solutions to this problem [29] use Bloom filters to build an approximate index in order to save space. Mantis uses the CQF to build an exact index and outperforms other solutions on several dimensions. For example, previous solutions had large false-positive rates; Mantis has none. Mantis also builds the index in about one tenth the time, performs queries about 100 times faster, and uses 20% less space than the best Bloom-filter-based system.

Streaming. In streaming, my works shows how to build a scalable, precise, and real-time event detection system that is not possible to achieve in a traditional streaming setting. It challenges the assumption that only in-RAM data structures can keep up with real-world streams. It shows that by using modern storage devices and building upon recent advances in external-memory dictionaries, we can design on-disk data structures that can process millions of stream events per second.

Real-time monitoring of high-rate data streams, with the goal of detecting and preventing malicious events, is a critical component of defense systems for cybersecurity as well as for physical systems, e.g., for water or power distribution. Accuracy (i.e., few false-positives and no false-negatives) and timeliness of event detection are essential to these systems [6]. Central to these applications is the Timely Event Detection (TED) problem in which given a stream $S = (s_1, ..., s_N)$ and a reporting threshold $T = \phi N$, where N is the size of the stream, we need to report all elements that occur more than T times in S soon after the T-th occurrence. To solve the TED problem one needs to count the number of occurrences of all distinct items in the stream with a tiny or no error. This requires a large amount of space ($\Omega(N)$ words) and is not solvable in the streaming model or via standard sampling-based approaches [6].

I developed leveled external-memory tables (LERTs) [25] to solve the TED problem. LERTs are write-optimized data structures and perform timely event detection by efficiently scaling to SSDs. LERTs achieve up to 11 Million events/sec insertion throughput while timely reporting malicious events. My work showed that we can match the performance of cache-latency-bound in-memory data structures without any false-negatives or -positives. It further shows how to extend write-optimized data structures, which have fast updates but relatively slow queries, to support fast standing queries for timely event detection.

Storage systems. In file systems, we showed how we can better organize data on disk using write-optimized dictionaries to speed up file system operations. We developed BetrFS [14, 32], the first in-kernel file system that uses B$^\varepsilon$-trees, an asymptotically optimal write-optimized dictionary. A B$^\varepsilon$-tree is a B-tree, augmented with per-node buffers. New items are inserted in the buffer of the root node and when a node's buffer becomes full, messages are moved from that node's buffer to one of its children's buffers.

BetrFS can match or outperform traditional file systems on almost every operation, some by an order of magnitude. I implemented zone trees in BetrFS that were a significant part of the BetrFS paper at FAST 2016 that got the Best paper award. Our earlier paper was Runner-up for the Best paper at FAST 2015. To further improve the performance of BetrFS, we worked on integrating AMQ data structures (i.e., the quotient filer) into the B$^\varepsilon$-tree. AMQ data structures help in filtering out unnecessary disk I/O requests that are otherwise performed for negative queries thereby improving the I/O performance of a B$^\varepsilon$-tree. In our experiments, we showed that, we can achieve 1.001 I/O requests per positive query and 0.04 I/O requests per negative query.

Future agenda. There are three major challenges—**scalability**, **adaptability**, and **changing hardware**—that we need to tackle to support future data analyses.

Our ability to generate, acquire, and store data has grown exponentially over the past decade, transforming fields such as biology, cosmology, drug discovery, etc. As a result, increasingly complex and large-scale analyses are employed to gain insights into this data. The **scalability** of these complex data analyses is going to be one of the biggest challenges in the next decade. To tackle the scalability challenge of future data analyses, I plan to work on building scalable parallel data structures in distributed-memory to enable data analyses to efficiently scale out to thousands of nodes in a high-performance computing (HPC) environment. The main research challenge here is to design parallel data structures that can minimize the communication across the nodes to achieve high performance in a distributed environment.

Another challenge is the **adaptability** of data structures to the change in the distribution of data or queries. The changes in the distribution at run time result in sub-optimal performance which can be fixed only by rebuilding data processing pipelines from scratch. This causes data analysts to spend a considerable portion of their time in building and maintaining these pipelines and much less time actually analyzing data and making informed decisions. My goal is to build adaptable data structures that can infer and adapt to the changes in the query or data distribution at run time and offer optimal performance with no human intervention. These adaptable data structures will allow data analysts to save time in maintaining and optimizing data processing pipelines and allow them to actually analyze data.

Continuously **changing hardware**, such as storage devices and processing units, is giving rise to new algorithmic paradigms and making traditional data structures sub-optimal. For example, at the start of grad school, I was designing data structures for CPU, DRAM, and hard disk drives (HDD). A few years later, I was working on new storage mediums such as flash drives (SSD) and shingled disks. Now, I am working on data structures for GPUs and persistent memory. My goal is to continue to advance the theory and practice of data structures in the light of new algorithmic paradigms to make them faster and smaller on modern hardware.

As a data structure researcher, I am excited to be at this juncture and work on building the next generation of data structures. My research is **vertically integrated**—from theoretical aspects of data structures to using them to build an in-kernel file system to understanding the practical challenges in building scalable data applications. My research

is also **inter-disciplinary** as I have built multiple large-scale data management tools across computational biology, cyber-security, and storage. My inter-disciplinary research experience and active collaborations make me the right candidate to take on this challenging research and succeed.

Following are a few short- and mid-term research goals.

- **Pushing the limits of hash-based data structures:** Today's filters and hash maps have a tradeoff between space and speed; even at moderate load factors (e.g., 50%-75% full), their performance degrades nontrivially. Due to poor performance at high load factors today's systems designers are forced to choose between speed and space usage. Unfortunately, for many applications, insertion and deletion performance when the data structure is nearly full is the only update performance that matters. Having a constant and high insertion/deletion throughput irrespective of the load factor will allow these applications to operate data structures at a high load factor and still achieve good update throughput. We can achieve this by redesigning the filers and hash tables based on a new algorithmic paradigm that is made possible by the ultra-wide vector operations supported by AVX-512. The idea is: reduce the problem to subproblems of size O($\log^r N$) bits, where r is a small constant, say, 1.5 or 2, and then apply vector operations to solve each subproblem in constant time. This paradigm may be applicable to other data structures.
- **Petabyte-scale data index for omics data:** Analysis of genomic, transcriptomic, and metagenomic data combined with other biological annotations at population scale promises to improve applications such as personalized medicine, population-level disease analysis, cancer remission rate prediction, and novel species detection. Although numerous studies have been performed over the past decade involving omics data, the ability to scale these studies to petabyte-scale data available today is still limited. My plan is to build a distributed, scalable, and updatable index for omics data at peta-byte scale so that we can index all of the omics data available today, quickly add new incoming data to the index, and gain critical insights for biological studies. We can achieve this by using the recent advancements in succinct and compact data structures and transforming them into dynamic data structures using efficient mergeability. I also plan to deploy this index as a public service available to scientists around the world. Prior genomic search tools, such as BLAST [5], have played an instrumental role in advancing fundamental research in bioinformatics and been cited over 70K times. I hope to have a similar or greater impact, since the new index can search through even more data than BLAST.
- **Scalable data structures for approximate similarity search:** Approximate similarity search methods such as approximate nearest neighbor (ANN) and locality-sensitive hashing (LSH) act as a fast and efficient proxy for similarity measures that have high computational complexity, e.g., edit distance. These proxy methods have been at the core of numerous tasks across databases, data mining, and bioinformatics. These methods determine low-distortion embeddings of the input data and then perform searches on these embeddings instead of the original data. However, given the size of data today it is getting challenging to even scale the approximate similarity search methods. I plan to work on developing new and innovative data structures for efficiently indexing and searching through large-scale collections of embeddings. My goal is to use an interleaved layout for storing these embeddings in memory to avoid unnecessary probes during a search operation and leverage ultra-wide vector operations supported by AVX-512 to speed up searches. Finally, I plan to build read-optimized data structures that can speed up searches when these indexes scale out of RAM to storage devices.

References

[1] F. Almodaresi, J. Khan, S. Madaminov, P. Pandey, M. Ferdman, R. Johnson, and R. Patro. An incrementally updatable and scalable system for large-scale sequence search using lsm trees. *bioRxiv*, 2021.

[2] F. Almodaresi, P. Pandey, M. Ferdman, R. Johnson, and R. Patro. An efficient, scalable and exact representation of high-dimensional color information enabled via de Bruijn graph search. In *International Conference on Research in Computational Molecular Biology (RECOMB)*, pages 1–18. Springer, 2019.

[3] F. Almodaresi, P. Pandey, M. Ferdman, R. Johnson, and R. Patro. An efficient, scalable, and exact representation of high-dimensional color information enabled using de bruijn graph search. *Journal of Computational Biology*, 27(4):485–499, 2020.

[4] F. Almodaresi, P. Pandey, and R. Patro. Rainbowfish: a succinct colored de Bruijn graph representation. In *17th International Workshop on Algorithms in Bioinformatics (WABI 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[5] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

[6] M. A. Bender, J. W. Berry, M. Farach-Colton, R. Johnson, T. M. Kroeger, P. Pandey, C. A. Phillips, and S. Singh. The online event-detection problem. *preprint arXiv:1812.09824*, 2018.

[7] M. A. Bender, A. Conway, M. Farach-Colton, W. Jannen, Y. Jiao, R. Johnson, E. Knorr, S. McAllister, N. Mukherjee, P. Pandey, et al. Small refinements to the dam can have big consequences for data-structure design. In *The 31st ACM on Symposium on Parallelism in Algorithms and Architectures*, pages 265–274. ACM, 2019.

[8] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[9] R. Chen, J. Shi, Y. Chen, B. Zang, H. Guan, and H. Chen. Powerlyra: Differentiated graph computation and partitioning on skewed graphs. *ACM Transactions on Parallel Computing (TOPC)*, 5(3):1–39, 2019.

[10] P. E. Compeau, P. A. Pevzner, and G. Tesler. How to apply de bruijn graphs to genome assembly. *Nature biotechnology*, 29(11):987–991, 2011.

[11] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[12] L. Dhulipala, G. E. Blelloch, and J. Shun. Low-latency graph streaming using compressed purely-functional trees. In *Proceedings of the 40th ACM SIGPLAN conference on programming language design and implementation*, pages 918–934, 2019.

[13] M. Goswami, D. Medjedovic, E. Mekic, and P. Pandey. Buffered count-min sketch on ssd: Theory and experiments. In *ESA*, 2018.

[14] W. Jannen, J. Yuan, Y. Zhan, A. Akshintala, J. Esmet, Y. Jiao, A. Mittal, P. Pandey, P. Reddy, L. Walsh, M. A. Bender, M. Farach-Colton, R. Johnson, B. C. Kuszmaul, and D. E. Porter. Betrfs: A right-optimized write-optimized file system. In J. Schindler and E. Zadok, editors, *Proceedings of the 13th USENIX Conference on File and Storage Technologies, (FAST), Santa Clara, CA, USA, February 16-19, 2015*, pages 301–315. USENIX Association, 2015.

[15] W. Jannen, J. Yuan, Y. Zhan, A. Akshintala, J. Esmet, Y. Jiao, A. Mittal, P. Pandey, P. Reddy, L. Walsh, et al. Betrfs: A right-optimized write-optimized file system. In *13th {USENIX} Conference on File and Storage Technologies ({FAST} 15)*, pages 301–315, 2015.

[16] W. Jannen, J. Yuan, Y. Zhan, A. Akshintala, J. Esmet, Y. Jiao, A. Mittal, P. Pandey, P. Reddy, L. Walsh, et al. Betrfs: Write-optimization in a kernel file system. *ACM Transactions on Storage (TOS)*, 11(4):18, 2015.

[17] G. Marçais, D. DeBlasio, P. Pandey, and C. Kingsford. Locality-sensitive hashing for the edit distance. *Bioinformatics*, 35(14):i127–i135, 07 2019.

[18] P. Pandey, F. Almodaresi, M. A. Bender, M. Ferdman, R. Johnson, and R. Patro. Mantis: A fast, small, and exact large-scale sequence-search index. In *Research in Computational Molecular Biology*, page 271, 2018.

[19] P. Pandey, F. Almodaresi, M. A. Bender, M. Ferdman, R. Johnson, and R. Patro. Mantis: A fast, small, and exact large-scale sequence-search index. *Cell systems*, 7(2):201–207, 2018.

[20] P. Pandey, M. A. Bender, R. Johnson, and R. Patro. debgr: an efficient and near-exact representation of the weighted de Bruijn graph. *Bioinformatics*, 33(14):i133–i141, 2017.

[21] P. Pandey, M. A. Bender, R. Johnson, and R. Patro. A general-purpose counting filter: Making every bit count. In *Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD)*, pages 775–787. ACM, 2017.

[22] P. Pandey, M. A. Bender, R. Johnson, and R. Patro. Squeakr: an exact and approximate k-mer counting system. *Bioinformatics*, 34(4):568–575, 2017.

[23] P. Pandey, A. Conway, J. Durie, M. A. Bender, M. Farach-Colton, and R. Johnson. Vector quotient filters: Overcoming the time/space trade-off in filter design. In *Proceedings of the 2021 International Conference on Management of Data*, pages 1386–1399, 2021.

[24] P. Pandey, Y. Gao, and C. Kingsford. Variantstore: an index for large-scale genomic variant search. *Genome biology*, 22(1):1–25, 2021.

[25] P. Pandey, S. Singh, M. A. Bender, J. W. Berry, M. Farach-Colton, R. Johnson, T. M. Kroeger, and C. A. Phillips. Timely reporting of heavy hitters using external memory. In *Proceedings of the 2020 ACM International Conference on Management of Data (SIGMOD)*. ACM, 2020.

[26] P. Pandey, B. Wheatman, H. Xu, and A. Buluc. Terrace: A hierarchical graph container for skewed dynamic graphs. In *Proceedings of the 2021 International Conference on Management of Data*, pages 1372–1385, 2021.

[27] P. A. Pevzner, H. Tang, and M. S. Waterman. An eulerian path approach to dna fragment assembly. *Proceedings of the national academy of sciences*, 98(17):9748–9753, 2001.

[28] J. Shun and G. E. Blelloch. Ligra: a lightweight graph processing framework for shared memory. In *Proceedings of the 18th ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 135–146, 2013.

[29] B. Solomon and C. Kingsford. Improved search of large transcriptomic sequencing databases using split sequence bloom trees. In S. C. Sahinalp, editor, *Research in Computational Molecular Biology - 21st Annual International Conference, (RECOMB), Hong Kong, China, May 3-7, 2017, Proceedings*, volume 10229 of *Lecture Notes in Computer Science*, pages 257–271, 2017.

[30] J. Yuan, Y. Zhan, W. Jannen, P. Pandey, A. Akshintala, K. Chandnani, P. Deo, Z. Kasheff, L. Walsh, M. Bender, et al. Optimizing every operation in a write-optimized file system. In *14th {USENIX} Conference on File and Storage Technologies ({FAST} 16)*, pages 1–14, 2016.

[31] J. Yuan, Y. Zhan, W. Jannen, P. Pandey, A. Akshintala, K. Chandnani, P. Deo, Z. Kasheff, L. Walsh, M. A. Bender, et al. Writes wrought right, and other adventures in file system optimization. *ACM Transactions on Storage (TOS)*, 13(1):3, 2017.

[32] J. Yuan, Y. Zhan, W. Jannen, P. Pandey, A. Akshintala, K. Chandnani, P. Deo, Z. Kasheff, L. Walsh, M. A. Bender, M. Farach-Colton, R. Johnson, B. C. Kuszmaul, and D. E. Porter. Optimizing every operation in a write-optimized file system. In A. D. Brown and F. I. Popovici, editors, *14th USENIX Conference on File and Storage Technologies, (FAST), Santa Clara, CA, USA, February 22-25, 2016.*, pages 1–14. USENIX Association, 2016.

[33] Y. Zhang, V. Kiriansky, C. Mendis, S. Amarasinghe, and M. Zaharia. Making caches work for graph analytics. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 293–302. IEEE, 2017.

[34] X. Zhu, W. Han, and W. Chen. Gridgraph: Large-scale graph processing on a single machine using 2-level hierarchical partitioning. In *2015 {USENIX} Annual Technical Conference ({USENIX}{ATC} 15)*, pages 375–386, 2015.