Research Statement

Prashant Pandey
ppandey2@cs.cmu.edu

I design and build theoretically well-founded data structures for big data problems in computational biology, storage, and streaming. My work is having a transformative impact on computational biology, because I am essentially rebuilding the entire genomic and transcriptomic analysis tool chain around data structures of my design, and I am obtaining significant gains in speed, accuracy, space-efficiency, and simplicity. This work is described in four flagship conferences (SIGMOD, IMSB, RECOMB, WABI) [1, 2, 17, 20, 21] and two top journal papers (Cell Systems, Bioinformatics) [18, 22]. It is in use in at least ten top labs around the world from both academia and industry.

In storage systems, my work shows how to use modern data structures to overcome decades-old tradeoffs in file-system design, yielding orders-of-magnitude performance gains. This work is described in two top-tier conferences (FAST, SPAA) [4, 13, 25] and two journal papers (ToS) [14, 26], including a Best Paper Award at FAST 2016. In streaming, my work bridges the world of external-memory and streaming algorithms. Furthermore, it shows how we can use external memory for problems that have traditionally been analyzed in the streaming setting to enable solutions that can scale beyond the provable limits of fast RAM. This is an ongoing project and some preliminary results are published at ESA and arXiv [5, 11].

My research has pursued two basic strategies for managing big data: shrink it, and organize it. For applications in computational biology and databases that can afford a small number of errors in results, I shrink data in favor of space-efficiency and speed. On the other hand, for storage systems, where accuracy is critical, I organize data in a disk friendly way to efficiently perform disk accesses and achieve better performance.

Computational biology. In computational biology, we showed how to use recent theoretical advances in compact and succinct data structures to shrink genomic and transcriptomic data down to the size where it can be processed on a laptop. I first developed Squeakr [22] to solve the k-mer counting problem. In k-mer counting, we count the number of occurrences of each k-mer (a length-k substring) in a sequencing dataset. In particular, k-mer counting is used to weed out erroneous data caused by sequencing errors. k-mer counting is one of the most common and arguably one of the simplest preliminary step in many bioinformatics analyses. In particular, k-mer counting is used to weed out erroneous data caused by sequencing errors. The number of existing papers on this problem suggest, however, that efficient execution of this task, with reasonable memory use, is far from trivial.

Squeakr uses the counting quotient filter (CQF) [21], a counting data structure we developed for fast approximate and exact counting, to count k-mers. Squeakr being smaller in size 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.

I then developed deBGR [20], which efficiently represents de Bruijn graphs which are at the heart of many sequence analyses. In computational biology, the de Bruijn graph is used to represent k-mer content in sequencing datasets. In the de Bruijn graph, a k-mer represents an edge connecting its two $(k-1)$-mer (nodes) substrings. Despite the computational benefits that the de Bruijn graph provides, the graph still tends to require a substantial amount of memory for large data sets making many analyses slow or even impossible

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 other 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 many de Bruijn graph analyses in RAM.

I also developed a large-scale search index Mantis [17], which attempts to solve the fundamental problem of large-scale sequence-search over tons of raw sequencing datasets. There's a huge amount of information available in raw sequencing (unassembled) data which gets lost during the assembly process. As a result, assembled data is hugely lossy. For example, a lot of variability information is lost during assembly. 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.

Mantis enables quickly searching through thousands of raw datasets, constituting terabytes of data. The ability to perform sequence searches is so essential that BLAST [3], an earlier tool for sequence-search, is one of the most cited papers in the entire 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. Prior solutions to this problem [23] use Bloom filters (a compact and approximate set representation data structure) 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 eighth the time, performs queries about 100 times faster, and uses 20% less space than the best Bloom-filter-based system.

Quotient filter. Much of my computational biology work builds on my research into Approximate Membership Query (AMQ) data structures. AMQs such as the Bloom filter [7], are almost five decades old and have been a workhorse in numerous applications in databases, storage systems, networks, computational biology, and other domains. However, many applications must 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. The CQF is a feature rich and practical data structure which is backed by strong theoretical results for space requirement and insert/query complexity and offers an order-of-magnitude faster operations than Bloom filters. The CQF is small and fast, supports counting (even on skewed data sets), and has features that other AMQs lack, e.g., deletions, resizing, and scaling out-of-RAM. Based on my research, several (at least ten) top teams around the world have replaced Bloom filters in their code with counting quotient filters.

Succinct data structures take space close to the information-theoretic lower bound (i.e., $z + o(z)$, where z is the lower bound) and use rank-and-select bit vector operations for operations. Similar to succinct data structures [10, 12] the CQF is also based on rank-and-select operations. Though, asymptotically optimal, many succinct data structures have poor performance due to the constants involved in bit-vector rank-and-select implementations [24]. However, in CQF, we use new x86 bit-manipulation instructions to speed up rank-and-select operations and our implementation can also be useful for other rank-and-select-based data structures which were not practical before [19].

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 [13, 25], the first in-kernel file system that uses B$^\varepsilon$-tree, an asymptotically optimal write-optimized dictionary. A B$^\varepsilon$-tree is a B-tree, augmented with pernode 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 nodes 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 Runners-up at FAST 2015. To further improve the performance of BetrFS we worked on integrating AMQ data structures (i.e., the quotient filer) in B$^\varepsilon$-tree. In this work, we showed that by integrating an AMQ data structure we can improve 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 non-repeated query and 0.04 I/O requests per negative query.

Streaming. Many real-world monitoring systems, e.g., defense systems for cyber security and power and water distribution systems, demand event detection where all events must be reported (no false negatives), in a timely manner, and a low reporting threshold (i.e., items with relatively small counts compared to the size of the stream). This problem is called as the online event-detection problem (OEDP). As a result, to solve OEDP 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 [5]. Our work shows that by using write-optimized data structures and cutting-edge hardware, we can solve the OEDP in a timely manner by devising solutions that can scale out-of-RAM to SSDs. 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 standing queries. This is an ongoing project in collaboration with researchers from Sandia National Labs.

Industry experience. In addition of my academic research I also had the opportunity to explore challenging problems in industry via internships. During my internships I worked on problems different from my own research. This helped me in two ways. First, in getting a better understanding of domains other than my own research. Second, learning on how to apply principles from one domain to solve problems in a different one. While interning at Intel Labs, I worked on an encrypted FUSE file system using Intel SGX [15]. I also developed new SGX instructions to fork processes securely from a process already running inside an SGX environment. At Google, I designed and implemented an extension to the ext4 file system for cryptographically ensuring file integrity. Google is currently working to integrate this extension into Android and the mainline Linux kernel. While at Google, I also worked on optimizing the memory usage of core in-memory data structures of Spanner, Google's geo-distributed big database. The work I did during the internship is being used in the Google Spanner's production systems.

Future work. I plan to target three primary lines of work in my future research. First, I will to continue to scale Mantis to population scale datasets and also work on indexing other large-scale datasets like population variation dataset. Second, I hope to apply the idea of trading accuracy for space for representing extreme-scale graphs in other domains like social networks and natural languages. Third, I plan to continue my research into AMQ data structures making them more efficient in terms of space and performance and also work adaptive AMQs.

I hope to scale Mantis to petabytes of data and deploy it as a public service available to scientists around the world. Prior genomic search tools, such as BLAST [3], has played an instrumental role in advancing fundamental research in bioinformatics and evidently been cited over 70K times. I hope to have a similar or greater impact with Mantis, since Mantis can search through even more data than BLAST. I also hope to continue shrinking computational biology data so that we can start performing analyses on phones or other devices that can be carried into the field, where internet or even laptops may not be available or practical.

I plan to apply the basic strategy behind deBGR – using invariants from the underlying graph to create a compact representation – to other big data graph problems, such as in social networks, natural languages,

etc. Shrinking these graphs can help accelerate many applications in machine learning and natural language processing which perform analysis on extreme-scale graphs.

I plan to work on an efficient AMQ data structure implementation which offers near-constant insertion throughput irrespective of the load factor of the data structure. Currently, insertion throughput of all state-of-the-art AMQs (quotient filter, cuckoo filter [9], morton filter [8]) degrades with increasing load factor. Many applications in storage, databases, and streaming operate these AMQs at a low load factor to achive higher insertion throughput and at the cost of sub-optimal space usage. Having a constant insertion throughput irrespective of the load factor will allow these applications to operate AMQs at a high load factor and still achive good insertion throughput.

I also plan to work on the implementation of adaptive AMQ data structures [6, 16] and replacing traditional AMQs with adaptive AMQs in applications. An adaptive AMQ data structure guarantees a fixed error rate irrespective of the query distribution unlike traditional AMQs which only guarantee a given error rate for uniform-random query distribution. Currently, there are no efficient adaptive AMQ implementations. Also, replacing traditional AMQs with the adaptive ones in applications can open several avenues of optimizations and would also involve rethinking the design of applications around adaptive AMQs.

References

[1] 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. *(To appear at RECOMB 2019)*.

[2] F. Almodaresi, P. Pandey, and R. Patro. Rainbowfish: A succinct colored de bruijn graph representation. In R. Schwartz and K. Reinert, editors, *17th International Workshop on Algorithms in Bioinformatics, (WABI) , August 21-23, 2017, Boston, MA, USA*, volume 88 of *LIPIcs*, pages 18:1–18:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

[3] 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.

[4] M. Bender, A. Conway, M. Farach-Colton, W. Jannen, Y. Jiao, R. Johnson, E. Knorr, S. McAllister, N. Mukherjee, P. Pandey, D. Porter, J. Yuan, and Y. Zhan. The dictionary problem, optimal searching, and asymptotic distortions of the dam. *(To appear at SPAA 2019)*.

[5] 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.

[6] M. A. Bender, M. Farach-Colton, M. Goswami, R. Johnson, S. McCauley, and S. Singh. Bloom filters, adaptivity, and the dictionary problem. In M. Thorup, editor, *59th IEEE Annual Symposium on Foundations of Computer Science, (FOCS) , Paris, France, October 7-9, 2018*, pages 182–193. IEEE Computer Society, 2018.

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

[8] A. Breslow and N. Jayasena. Morton filters: Faster, space-efficient cuckoo filters via biasing, compression, and decoupled logical sparsity. *PVLDB*, 11(9):1041–1055, 2018.

[9] B. Fan, D. G. Andersen, M. Kaminsky, and M. Mitzenmacher. Cuckoo filter: Practically better than bloom. In A. Seneviratne, C. Diot, J. Kurose, A. Chaintreau, and L. Rizzo, editors, *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies, (CoNEXT), Sydney, Australia, December 2-5, 2014*, pages 75–88. ACM, 2014.

[10] P. Ferragina and G. Manzini. Opportunistic data structures with applications. In *41st Annual Symposium on Foundations of Computer Science (FOCS), 2000.*, pages 390–398. IEEE, 2000.

[11] M. Goswami, D. Medjedovic, E. Mekic, and P. Pandey. Buffered count-min sketch on SSD: theory and experiments. In Y. Azar, H. Bast, and G. Herman, editors, *26th Annual European Symposium on Algorithms, (ESA), August 20-22, 2018, Helsinki, Finland*, volume 112 of *LIPIcs*, pages 41:1–41:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

[12] R. Grossi and J. S. Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM Journal on Computing*, 35(2):378–407, 2005.

[13] 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.

[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: Write-optimization in a kernel file system. *Transactions on Storage (TOS)*, 11(4):18:1–18:29, 2015.

[15] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar. Innovative instructions and software model for isolated execution. In R. B. Lee and W. Shi, editors, *The Second Workshop on Hardware and Architectural Support for Security and Privacy (HASP), Tel-Aviv, Israel, June 23-24, 2013*, page 10. ACM, 2013.

[16] M. Mitzenmacher, S. Pontarelli, and P. Reviriego. Adaptive cuckoo filters. In R. Pagh and S. Venkatasubramanian, editors, *Proceedings of the Twentieth Workshop on Algorithm Engineering and Experiments, (ALENEX), New Orleans, LA, USA, January 7-8, 2018.*, pages 36–47. SIAM, 2018.

[17] 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 B. J. Raphael, editor, *Research in Computational Molecular Biology - 22nd Annual International Conference, (RECOMB), Paris, France, April 21-24, 2018, Proceedings*, volume 10812 of *Lecture Notes in Computer Science*, pages 271–273. Springer, 2018.

[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. *Cell systems*, 7(2):201–207, 2018.

[19] P. Pandey, M. A. Bender, and R. Johnson. A fast x86 implementation of select. *arXiv preprint arXiv:1706.00990*, 2017.

[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 S. Salihoglu, W. Zhou, R. Chirkova, J. Yang, and D. Suciu, editors, *Proceedings of the 2017 ACM International Conference on Management of Data, (SIGMOD), Chicago, IL, USA, May 14-19, 2017*, 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, 2018.

[23] 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.

[24] S. Vigna. Broadword implementation of rank/select queries. In *International Workshop on Experimental and Efficient Algorithms*, pages 154–168. Springer, 2008.

[25] 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.

[26] 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. Writes wrought right, and other adventures in file system optimization. *Transactions on Storage (TOS)*, 13(1):3:1–3:26, 2017.