Research Statement and Agenda

Prashant Pandey
ppandey@berkeley.edu

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. For example, numerous biological questions can be answered by performing sequence-level searches over petabytes of publicly-available raw sequencing data stored in the sequence read archive (SRA) [25]. Defense systems for cybersecurity and physical systems for water and power distribution require monitoring a high-speed stream (10+ Million events/sec) over long periods of time and need to detect and report anomalous events in real time [7, 24]. Given the scale of data analyses now underway and planned in the near future it is ever so important to build efficient pipelines to process, store, and make data available for these complex analyses.

My research has focused on designing and building tools backed by theoretically well-founded data structures for large-scale data management problems across computational biology, stream processing, and storage. I have pursued two basic strategies for large-scale data management: shrink it and organize it. For applications that can afford a small, bounded fraction of errors in results, I lossily shrink data in favor of space and speed. On the other hand, for applications where accuracy is critical, I organize data based on the underlying storage to achieve better performance.

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 eight papers in three flagship conferences (**RECOMB, ISMB, WABI**) and three journals (**Cell Systems, Bioinformatics, JCB**) [1–3, 16–18, 20, 21]. In stream processing, my work builds a bridge between the worlds of external-memory and streaming algorithms. It 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. This work is described in four papers in two top-tier conferences (**SIGMOD, ESA**) [5, 12, 19, 22]. 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 journal (**ToS**) including a Best Paper Award at FAST 2016 [6, 14, 15, 26, 27].

Memory and resource-efficient 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) [19] 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) [10] 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 [12], 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.

Computational biology. In computational biology, I showed how to use recent theoretical advances in compact and succinct data structures to build efficient indexes for large-scale genomic and transcriptomic data.

I first developed Squeakr [20] 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 [19], 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 [18], which efficiently represents weighted de Bruijn graphs which are at the heart of many sequence analyses [9, 23]. 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, 2, 17], 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 [4], 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 [17, 25]. Prior solutions to this problem [25] 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 work 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 [5]. 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 [5].

I developed leveled external-memory tables (LERTs) [22] 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 [13, 28], 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 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 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. The scalability of complex data analyses is going to be one of the biggest challenges in the next decade. At the same time, continuously changing hardware, such as storage devices and CPUs, is giving rise to new algorithmic paradigms. I believe these challenges are also opportunities to rethink the design of existing data structures to take full advantage of modern hardware and rebuild data management systems to efficiently support future data analyses. I plan to build next generation data structures that are suited to modern hardware. I would further use these advancements to develop data management infrastructure to accelerate scientific analyses such as large-scale sequence-search in computational biology and streaming graph analytics in social networks and natural language processing.

- **Pushing the limits of filters and hash tables:** Todays filters have a tradeoff between space and speed; even at moderate load factors (e.g., 50%-75% full), their performance degrades nontrivially. For example, the insertion throughput in the cuckoo filter [11] drops $16\times$ when going from 10% full to 90% full and in the quotient filter [19] it drops $4\times$. Due to poor performance at high load factors todays systems designers are forced to chose between speed and space usage. Unfortunately, for many applications, insertion and deletion performance when the filter is nearly full is the only update performance that matters. For example, a database might maintain filters of its tables to estimate join sizes during query planning. A network caching system might maintain an in-memory filter of a large, on-disk cache. In both cases, it is not practical to rebuild the filter from scratch, so the filter must support inserts and deletes, and, for space efficiency, operate at a high load factor. Having a constant and high insertion/deletion throughput irrespective of the load factor will allow these applications to operate filters at a high load factor and still achieve good update throughput. We can achieve this by redesigning the filter 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, such as hash tables and succinct data structures, e.g., tries.
- **Population-scale genomic data index:** Population-scale genomic sequencing efforts produce genomic variation data from thousands of samples, such as the 1000 Genomes project, GTEx, and The Cancer Genome Atlas (TCGA). Analysis of genomic variants combined with phenotypic information of samples promises to improve applications such as personalized medicine, population-level disease analysis, and cancer remission rate prediction. Although numerous studies have been performed over the past decade involving genomic variation, the ability to scale these studies to large-scale data available today and in the near future is still limited. Using my expertise in compact data structures and large-scale indexing, I plan to build an efficient variant index that can be iteratively scaled to thousands of samples and enable medical and scientific application quickly perform complex analyses and answer high-level biological questions. I also plan 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 [4], 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 with Mantis, since Mantis can search through even more data than BLAST.
- **Accelerating graph analytics on skewed streaming graphs:** Various applications model problems as streaming graphs and need to quickly apply a stream of updates and run algorithms on the updated graph. Furthermore, many dynamic real-world graphs, such as social networks, follow a skewed distribution of vertex degrees, where there are a few high-degree vertices and many low-degree vertices. Existing static graph-processing systems achieve high performance and low space usage by exploiting graph skewness via pre-processing a cache-efficient graph partitioning based on vertex degree. 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 plan to use techniques like dynamic

hierarchical partitioning of the vertexes based on their degrees and write-optimization to build a cache-efficient dynamic graph representation and enable fast graph analytics. The dynamic partitioning approach can be further combined with iterative graph algorithms that perform computations on streaming graphs to build highly-optimized streaming graph systems. I also plan to shrink large-scale graphs, such as in social networks and natural languages, using a probabilistic representation in favor of space. Shrinking these graphs will help accelerate applications in machine learning and natural language processing that perform analysis on extreme-scale graphs.

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. In *International Conference on Research in Computational Molecular Biology (RECOMB)*, pages 1–18. Springer, 2019.

[2] 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.

[3] 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.

[4] 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.

[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, 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.

[7] J. Berry, R. D. Carr, W. E. Hart, V. J. Leung, C. A. Phillips, and J.-P. Watson. Designing contamination warning systems for municipal water networks using imperfect sensors. *Journal of Water Resources Planning and Management*, 135, 2009.

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

[9] 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.

[10] 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.

[11] 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.

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

[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, 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.

[15] 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.

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

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

[18] 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.

[19] 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.

[20] 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.

[21] P. Pandey, Y. Gao, and C. Kingsford. Variantstore: A large-scale genomic variant search index. *bioRxiv*, 2019.

[22] 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.

[23] 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.

[24] S. Raza, L. Wallgren, and T. Voigt. Svelte: Real-time intrusion detection in the internet of things. *Ad Hoc Networks*, 11(8):2661–2674, 2013.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

Teaching Statement

Prashant Pandey
ppandey@berkeley.edu

I believe teaching and mentorship are keys to sustainable research. I have had several chances to teach and mentor during my grad school and postdoc. Both teaching and mentoring have time and again helped me get a different perspective towards research and in turn, rejuvenated my interest in learning new things.

Teaching. I was a teaching assistant (TA) for four different courses during my grad school. I taught lab sessions for "Intro to Programming" for undergrads in the first semester of grad school. In the second semester, I again taught lab sessions for "Advanced Programming" for undergrads. In lab sessions, my aim was to first teach students how to think about converting a simple data structure or an existing algorithm into pseudocode. Once they had the pseudocode, I taught them programming language concepts by live-coding using one of the pseudocodes. This way students learned about all the common programming errors and ensuing typos/bugs. This exercise also made them view me as a human computer scientist and not a mere dispenser of facts.

Later in my fifth semester, I TA'ed for two grad courses, "Asynchronous Systems" and "Analysis of Algorithms" — a class consisting of 180 students. I helped co-design homework and exam problems with the professor and other TAs in the algorithm course. For the "Asynchronous Systems" course, I co-designed semester-wide assignments and the final project. I also judged the final project demos. TA'ing for grad courses was quite a different challenge compared to undergrad lab sessions. It taught me how to structure a semester-long grad level course and how to come up with assignments/homework problems and keeping them in sync with the teaching material in the class.

The first semester was my first time teaching a course and at the same time to a very diverse set of students. I wanted to improve my communication and teaching skills so I enrolled myself in a "Communication Skills for Teaching" course in the humanities department at the university. This course taught me a lot of subtle things about teaching. For example, when a student asks a question it is better to repeat the question for everyone before answering it, how to maintain eye contact with students at all times, how to use examples to explain tricky concepts. The professor in the communication skills course also made us deliver a lecture on a topic in our field to students from other departments. Teaching computer science concepts to students from chemical, economics, and humanities department was a great learning experience. After teaching in the first semester and communication skills course, teaching lab sessions in the second semester was a lot of fun.

Mentoring. I mentored four students across four different projects during my grad school and postdoc. Two of them were junior Ph.D. students, one master's student, and one undergrad. I had to adapt my mentoring approach for different students. For some students, I took a more hands off approach and only helped them when they were stuck. For others, a more hands on approach worked better which included weekly updates and adhoc meetings to resolve issues quickly.

My aim while mentoring students is to make sure that they keep making progress and not go down a dead end path because of the lack of timely guidance. I also like to make sure that junior students should feel as much a part of the project as anyone else. To achieve this it is important to encourage them to take part in discussions and raise their concerns in the meetings.

While mentoring it is also important to understand how to guide students to the solution rather than just giving the solution. Guiding them to the solution can be time-consuming initially but it is always the better option in the long run. It really helps students learn and at the same time gain the much-needed confidence they need early in a research project.

Teaching philosophy. My attitude in teaching is to emphasize logic over facts. It is easier to fix the problem when the logic is correct but facts are wrong. However, it is harder to correct the logic once it is understood wrongly. Therefore, whenever I am teaching a new concept or an algorithm I always spend enough time to give the necessary intuition behind the concept/algorithm rather than digging straight into the complexity and implementation.

I also believe in learning by making errors. In order to get to the solution to a given problem first learning about the solutions that do not work and why helps our ability to think critically about the correct solution. This also helps students develop a deep computer science thinking mindset. For example, in the lab sessions, whenever we started to think about a new algorithm and how to implement it in a language I always asked students to first come up with the most naive solutions. Once everyone understood the naive solution we iteratively fixed it to reach the optimal solution.

These iterations of getting to the correct solution also teaches students the generality of the tricks and optimizations so that if they encounter a different problem they should still be able to apply those tricks and optimizations. Many students who took “Intro to Programming” with me in the first semester also took “Advanced Programming” in the second semester. In the “Advanced Programming” course, I could clearly see those students being able to easily argue about complex data structures and algorithms using the same tricks they learned in the first semester.

New courses. Given that my research interests lie at the intersection of systems and algorithms and I also work in computational biology I would be able to teach a few courses across domains. I would be able to teach general courses on data structures and algorithms, databases, operating systems, and programming. I can also teach an “Intro to Algorithms in Computational Biology” course. All these courses can be designed for both undergrad and grad level.

I hope to design a course emphasizing the rationale behind hashing and approximate data structures, like Bloom filters and quotient filters, and how to use them in building high-performant large-scale indexing and storage systems. This course will look at applications from different domains like storage, databases, streaming, and computational biology where both hashing and approximate data structures are heavily used. I also hope to design a course aimed at learning various data structures and algorithms that are used in systems and how to think about systems-specific problems in an algorithmic way.

I also envision a paper reading course designed for specific research areas, e.g., data structures and algorithms in systems, compact data structures in computational biology, etc. In the course, we will pick top research papers that have steered the field in a new direction. We will go over the important concepts in the paper and discuss how/why the authors of the paper did and did not do things. A paper reading course is a great way for junior students to get to know the field better and decide whether they want to do research in the area. At the same time, this will also be a good fit for senior students, who can also lead the discussion, to understand the rationale behind the seminal papers in the field.

Diversity Statement

Prashant Pandey
ppandey@berkeley.edu

Diversity is a boon to academia. It acts like a catalyst to nurture groundbreaking ideas that are needed to solve the next generation problems. However, it is also an inherently complex issue that needs to be dealt with the right attitude to realize its full potential. I am glad that it is getting the much needed attention. I would like to share my own views on diversity and what I hope to do to pursue a diverse and inclusive academic environment.

My perspective on diversity is shaped by my own background: my dad used to work in the Air Force and while I was growing up we use to move every two to three years. This made me live and get education in different parts of India which is affluent with diversity in terms of spoken language, culture, food, religion, and race. I changed multiple schools, made several friends, and was taught by teachers coming from a very diverse set of backgrounds. I understand, from experience, how adjusting to new cultures is intimidating and difficult, and how it can take a long time to dispel the feeling of being an outsider.

I believe that diversity comes in multiple forms, including but limited to, race, gender, social class, and sexual orientation. I want to draw from my own experience and talk about another kind of diversity, physical capability. While growing up I had a massive stage fear and use to stammer at times due to it. I used to be teased and made fun of by other kids that exacerbated the problem. I was often overlooked whenever a student had to be picked to represent the school for a debate or a recitation competition. However, one of my teachers played an instrumental role in helping me overcome my problem. He saw some potential in me and gave me the initial opportunities to go on stage and overcome my fear. I did stammer the first few times but soon I started to feel comfortable and went on to win numerous city and state level debate competitions. My experience has taught me two things. First, not being able to look or talk in the same way as others around you does not make you any less from others. It just means that you are different. It is important for us to accept the diversity as a normal and embrace it. Second, it is extremely important for the prominent members of the community to have the right attitude towards diversity. It sets the right example for the upcoming and junior members to follow.

I hope to create an inclusive environment for upcoming members of the computer science community so that they can think freely and excel in research. It is very important to provide the necessary guidance and opportunities to the underprivileged section of our community. I plan to engage with students from underrepresented backgrounds on a regular basis and will try and understand the challenges they face in day-to-day life. Understanding their challenges is the first step towards helping them to alleviate those challenges and bringing them closer to science.

I also plan to create an affirming research environment to improve diversity in computer science and science in general. Affirmation goes well beyond just inclusion; it entails recognizing the support, consideration, and encouragement that junior members should be receiving. Computer science is a field which sees success in bringing people from different fields across science and solving next generation computation problems. Creating an affirming environment is more important in computer science than any other field in science.