The University of Western Ontario
Department of Computer Science

Approximate Membership Queries in Sports Analytics

Group Members:
Randeep Bhalla (251259483)
Sahibjot Boyal (251164782)
Gurshawn Lehal (251159080)
Imesh Nimsitha (251171614)

Databases II (Advanced Databases)

Project Report

December 14, 2024

Contents

1 Introduction

1.1 Overview

Approximate Membership Query (AMQ) structures are specialized data structures designed to handle membership tests with high efficiency and minimal memory usage. They provide a probabilistic means of determining whether an element is a member of a set, allowing a small probability of false positives but guaranteeing no false negatives. This makes them effective in large-scale data management tasks where exact matches would be computationally expensive.

The three AMQ structures explored in this report are **Bloom Filters**, **Cuckoo Filters**, and **Vacuum Filters**, each offering unique trade-offs in terms of memory efficiency, query speed, and operational flexibility. While Bloom Filters are well-known for their simplicity and widespread use, Cuckoo Filters introduce dynamic resizing and deletion support. Vacuum Filters further optimize space usage and query performance through advanced space management techniques.

This report evaluates these AMQ structures in both theoretical and practical contexts. A key aspect of this evaluation is extending their application to a previously unexplored domain: **basketball analytics**. By applying AMQ structures to a large dataset of basketball player statistics, this project explores how these filters can be used to efficiently manage and query sports data, a field that demands fast, scalable, and memory-efficient data management solutions.

1.2 Significance

AMQ structures have become essential tools in modern data-intensive systems due to their ability to handle large datasets with limited memory footprints. Traditionally, they are employed in fields such as:

- *Database Management Systems (DBMSs)*: For indexing, caching, and query optimization.
- *Network Security*: For identifying known malicious patterns in real-time packet filtering.
- *Data Storage and Caching Systems*: To minimize storage costs while ensuring quick lookups.

Despite these common use cases, their potential in sports analytics remains underexplored. This project applies AMQs to the domain of basketball statistics, where datasets are large, dynamic, and query-intensive. Common queries in this context include determining whether a player with specific performance metrics exists in the dataset, or quickly retrieving subsets of players meeting certain statistical criteria (e.g. finding players with 20+ points per game). For simplicity, even determining whether a player exists in the dataset is a basic query that can become more efficient using AMQ structures. The significance of this research lies in:

- *Scalability*: Demonstrating how AMQs can handle a continuously growing dataset of player statistics.
- *Query Efficiency*: Showing that AMQs can reduce query response times, crucial for real-time analytics.
- *Resource Optimization*: Minimizing the memory required to store and retrieve sports data effectively.

By exploring the practical application of AMQ structures in sports analytics, this project highlights their adaptability and utility in managing complex, large-scale datasets beyond traditional database applications. The findings from this investigation contribute to advancing practices in data management and extending the scope of AMQs into new and impactful real-world applications.

2 Literature Review

2.1 Bloom Filters

Bloom Filters are probabilistic data structures that use multiple hash functions to map elements to a fixed-size bit array. To check for membership, the same hash functions are applied, and the corresponding bits are checked. If all bits are set to 1, the element is considered present; otherwise, it is not.

2.1.1 Strengths

- *Memory Efficiency*: The fixed-size bit array ensures low memory usage, making Bloom Filters suitable for scenarios where memory resources are constrained.
- *Straightforward Implementation*: Bloom Filters are conceptually simple and rely on fundamental operations such as hashing and bit manipulation. Their implementation involves initializing a bit array, applying hash functions, and setting or checking bits.

2.1.2 Limitations

- *Fixed Size*: The inability to resize the array dynamically can result in inefficiencies if the initial size is poorly estimated.
- *False Positives*: While false negatives are avoided, false positives become more likely as the dataset grows relative to the fixed bit array size (but configurable via the number of hash functions).

2.2 Cuckoo Filters

Cuckoo Filters extend Bloom Filters with dynamic resizing and deletion capabilities. Instead of a bit array, they store fingerprints of elements in hash table buckets. If a bucket is full during an insertion, an eviction mechanism relocates existing fingerprints to alternate locations, maintaining efficient space utilization.

2.2.1 Strengths

- *Deletion Support*: Elements can be deleted by removing their fingerprints, enabling adaptability for dynamic datasets.
- *Lower False-Positive Rates*: The use of fingerprints allows for a reduced probability of false positives compared to Bloom Filters with similar memory usage.

2.2.2 Limitations

- *Higher Memory Usage*: Fingerprint storage can increase memory consumption slightly compared to Bloom Filters.

2.3 Vacuum Filters

Vacuum Filters introduce advanced space management, offering enhanced performance and memory efficiency. They are particularly suited for large-scale, high-demand applications.

2.3.1 Strengths

- *Superior Memory Efficiency*: Vacuum Filters minimize the bits used per element, ensuring optimal resource usage for extensive datasets.
- *Fast Query Speed*: Their design prioritizes rapid membership checks, making them efficient for high query throughput.

2.3.2 Limitations

- *Complex Implementation*: Advanced algorithms increase the complexity of implementation and debugging.

3 Methodology

3.1 Tools and Technologies

To reproduce the results found in the literature, and to implement extensive applications on the AMQ structures, the following tools and technologies were used:

Python served as the programming language, offering an extensive standard library and third-party packages crucial for implementing data structures, algorithms, and computational tasks. For data manipulation and analysis, NumPy and Pandas were used. NumPy provided support for large, multi-dimensional arrays and matrices, along with a collection of high-level mathematical functions essential for performing the computations found in the filter algorithms. Pandas allowed for efficient reading, cleaning, merging, reshaping, and statistical analysis of structured data. To visualize research findings, Matplotlib was used to create charts that effectively represented the performance metrics. For the implementation of the Bloom and Cuckoo filters, the Hashlib module was used to generate hash values. The Bitarray module allowed for efficient bit array operations for membership testing in the Bloom filter. Ultimately, the research was based on the implementations of the AMQ structures found in the literature, which offered a foundational understanding of the field. The experiments were conducted using standard datasets of randomly generated integers, along with a predefined dataset of basketball player statistics.

The extensive application of AMQ structures used a comprehensive dataset of basketball player statistics from the 2023-24 NBA season, sourced from NBAStuffer, a reputable online website for basketball analytics. This dataset contained all 657 of the participating NBA players for that season, and their relevant traditional statistics such as points per game (PPG). The player data was carefully formatted into a CSV (Comma-Separated Values) file, which was then imported and used to populate the reproduced filters. This approach allowed for efficient data handling and analysis, thus providing a real-world application for testing the performance and accuracy of various AMQ structures.

3.2 Performance Metrics

The following metrics were used to conduct a statistical comparison of the three AMQ structures and their performance. In analyzing these metrics, each filter was evaluated to understand which one was optimal under various conditions, by considering accuracy, speed, and memory efficiency.

3.2.1 False Positive Rate (FPR)

The FPR was measured by checking for items in the *test* dataset that were not found in the *original* dataset using traditional lookup (exact matching) but were found in the filter using the filter's probabilistic lookup method. This metric was crucial for assessing the accuracy of the filter structures.

3.2.2 Query Speed

Query speed was measured as the total time taken to perform lookups on all items in the dataset divided by the dataset size (total number of items). This metric provided insights into the efficiency of each filter structure in handling lookup operations. In these experiments, the algorithm for calculating the FPR was used in conjunction with

calculating the query speed, as the FPR calculation used the filter's lookup method for all items in the dataset, which was an operation that the query speed benchmarked.

3.2.3 Memory Usage

Memory usage was measured in kilobytes (KB) to reflect the space efficiency of each filter, and varied depending on the filter type:

- *Bloom Filter*: Bit array memory usage
- *Cuckoo Filter*: Table memory usage
- *Vacuum Filter*: Table memory usage

3.3 Experimental Design

3.3.1 Reproduced Results from Literature

To reproduce the results found in the literature, the AMQ structures were implemented using the provided algorithms and evaluated using the outlined set of performance metrics. The procedure involved filter implementation, dataset preparation, filter population, and performance analysis.

The Bloom Filter, Cuckoo Filter, and Vacuum Filters were implemented based on the algorithms found in the literature. Each filter was coded as a separate Python class with methods for insertion, querying, (sometimes) deletion, and other auxiliary computations. The previously outlined tools and technologies were used for this implementation. To conduct the performance analysis, multiple datasets were generated to evaluate the effectiveness of the filters when handling varying dataset sizes. A small-sized dataset consisting of ten thousand items, a medium-sized dataset consisting of one hundred thousand items, and a large-sized dataset consisting of one million items were prepared. Each dataset was populated with random integers from 0 to 10,000,000 to ensure standardization. The AMQ filters were then populated by inserting all of the items found in the dataset. For all three datasets, a secondary dataset (the *test* dataset) was also created and populated with randomly generated integers, but was twice the size of the *original*. The *test* dataset may or may not have contained the integers found in the *original* dataset due to random generation, which was intentional and allowed for the testing of false positive memberships. Notably, an item that was found in the *test* dataset, that was not found in the *original* dataset (using exact matching lookup methods) but was found in the *original*'s AMQ filter (using the filter lookup method), was regarded as a false positive. Finally, the performance of the filters when handling the varying datasets was evaluated using the FPR, query speed, and memory usage performance metrics. These metrics were measured using a set of Python functions that queried the filters and kept track of attributes such as the number of false positives and the operation's calculation time. The measurements were then used for comparison with the results found in the literature. A chart visualization and a CSV file containing the table of resulting metrics were exported.

3.3.2 Extensive Application to Basketball Analytics

To evaluate the scalability and performance of AMQ structures with real-world applications, the analysis was extended to a dataset of basketball player statistics. The procedure involved dataset preparation, filter population, and performance analysis.

The dataset of basketball player statistics was imported and split to test the various AMQ filters. The player's name attribute was used as the key for creating and indexing the dataset. A new dataset (called the *original* dataset) was created consisting of 80% of the player records found in the imported dataset, and was used for filter population. A *test* dataset was created consisting of the remaining 20% of records, which was used for false positive membership testing. This 80/20 split allowed for efficient performance analysis and the evaluation of false positives. The reproduced AMQ filters from the previous experiments were then populated with the records found in the *original* dataset. By inserting all player records into these filters, the ability to handle a growing dataset of statistics was evaluated, thus testing the filter's scalability. For performance analysis, the same performance metrics used in the initial reproduction experiments were used to evaluate the filter's effectiveness in handling the basketball player dataset. Notably, the query speed and FPR metrics were crucial in evaluating a filter's performance under heavy query loads, by querying for all false positives in the entire dataset and measuring its resource allocation.

4 Results

4.1 Reproduced Results from Literature

To validate the theoretical characteristics of Bloom, Cuckoo, and Vacuum Filters observed in the literature, a performance analysis was conducted across the three key metrics. These experiments examined filter performance across three dataset sizes consisting of: 10,000, 100,000, and 1,000,000 items.

4.1.1 False Positive Rate (FPR)

The literature highlighted distinct FPR characteristics for each filter:

- *Bloom Filters*: Probabilistic false positives increasing with larger datasets.
- *Cuckoo Filters*: Lower false positive rates compared to Bloom Filters.
- *Vacuum Filters*: Minimal false positive probability.

The results from the analysis align similarly with these expectations, as shown in Table 1 (p. 9).

According to the data, Bloom Filters maintained a consistently low and stable false positive rate across the various dataset sizes. Meanwhile, the Cuckoo Filters measured increasing false positive rates as the dataset complexity grew. Finally, the Vacuum Filters measured near-zero false positive rates, confirming their advanced space management capabilities.

Dataset Size	Bloom FPR	Cuckoo FPR	Vacuum FPR
10,000	0.0162	0.06425	0.0
100,000	0.017015	0.117025	0.00002
1,000,000	0.013713	0.192228	0.0002015

Table 1: False Positive Rate Analysis [Reproduction]

4.1.2 Query Speed

The measurements for the query speed further validated the literature's performance claims about the AMQ structures, as shown in Table 2.

Dataset Size	Bloom Query Speed	Cuckoo Query Speed	Vacuum Query Speed
10,000	3.65e-06	4.86e-06	4.46e-06
100,000	1.34e-05	1.51e-05	1.50e-05
1,000,000	1.09e-04	1.14e-04	1.16e-04

Table 2: Query Speed Analysis (operations / second) [Reproduction]

These results aligned with the literature's claims about query efficiency. All three AMQ filters demonstrated similarly fast query speeds. Furthermore, the query speed scaled logarithmically with increasing dataset sizes. However, the Vacuum Filter maintained the most competitive query performance compared to the other two filters, and thus enabled the most rapid membership checks in large datasets.

4.1.3 Memory Usage

According to the literature, the Vacuum Filter was expected to maintain the lowest memory usage, while the Cuckoo Filter would use the most memory. Such expectations were slightly consistent with the observations from the analysis, as shown in Table 3.

Dataset Size	Bloom Memory	Cuckoo Memory	Vacuum Memory
10,000	12.29	83.18	25.43
100,000	122.15	782.21	240.71
1,000,000	1,220.78	8,250.71	2,258.27

Table 3: Memory Usage Analysis (KB) [Reproduction]

Across all three filters, there was minimal memory usage. According to the data, Bloom Filters demonstrated the most memory-efficient design. Meanwhile, the Cuckoo Filter showed higher memory consumption due to its fingerprint storage. Ultimately, the Vacuum Filter maintained a moderate memory usage, balancing the efficiency of the Bloom Filter and the performance of the Cuckoo Filter.

4.1.4 Visualization

To further understand the findings from the analysis, the resulting metrics were visualized as charts, as shown in Figure 1.

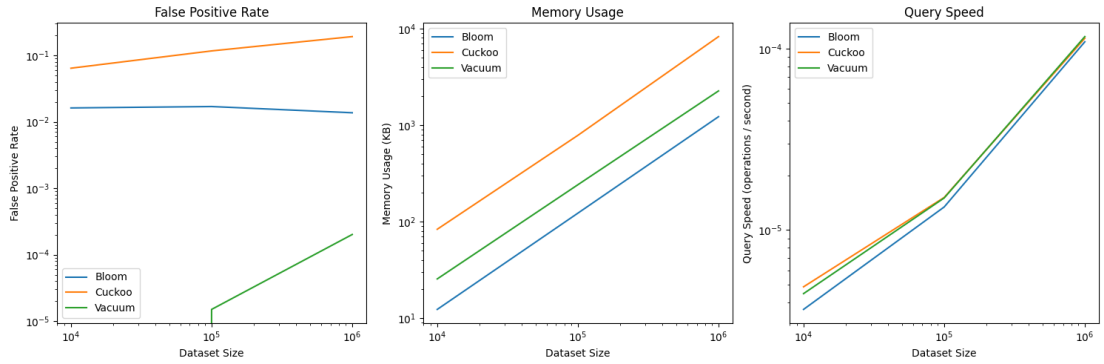

Figure 1: Visualization of Performance Metrics [Reproduction]

As visualized, the Vacuum Filter showed the best performance across the three metrics, and thus emerged as the most promising structure for large-scale, memory-efficient membership queries.

4.2 Extensive Application to Basketball Analytics

To demonstrate the applications of AMQ structures in sports analytics, a performance analysis was conducted on a predefined dataset of basketball player statistics using the same metrics for evaluating the implemented filters. The basketball dataset consisted of 657 player entries, and thus represented a compact sample size of statistics. This smaller dataset allowed for the examination of filter performance in a more nuanced context compared to the previous large-scale dataset analysis. This analysis provided the following insights into how these structures perform with a more focused, domain-specific dataset.

4.2.1 False Positive Rate (FPR)

Filter Type	False Positive Rate
Bloom	0.00758
Cuckoo	0.02273
Vacuum	0.0

Table 4: False Positive Rate Analysis [Basketball Dataset]

As shown in Table 4, the false positive rate results from the basketball dataset analysis mirrored the findings from the reproduction of the literature's results. The Bloom Filter maintained a low false positive probability, while the Cuckoo Filter showed a slightly

higher rate. Ultimately, the Vacuum Filter demonstrated the best accuracy, with zero false positives resulting from the dataset analysis.

4.2.2 Query Speed

Filter Type	Query Speed
Bloom	3.13e-06
Cuckoo	3.86e-06
Vacuum	3.70e-06

Table 5: Query Speed Analysis (operations / second) [Basketball Dataset]

As shown in Table 5, the measurements of the query speeds revealed near-identical performance across all three filters. Each structure showed extremely rapid lookup times when working with the compact dataset. There was minimal variation between the filter types, suggesting consistent efficiency across all of them. Overall, the performance overhead was negligible when dealing with basketball analytics.

4.2.3 Memory Usage

Filter Type	Memory Usage
Bloom	0.72
Cuckoo	4.68
Vacuum	1.62

Table 6: Memory Usage Analysis (KB) [Basketball Dataset]

As shown in Table 6, the memory usage of the three filters in the application of basketball analytics aligned with the memory usage from the reproduction of the literature's results. The Bloom Filter demonstrated the most compact memory footprint, while the Cuckoo Filter required the most memory. As expected, the Vacuum Filter offered a balanced approach from the other two filters.

5 Challenges

Throughout the implementation and performance analysis of the AMQ structures in sports analytics, several challenges emerged that tested the validity of the research methodology and computational approaches.

5.1 Performance Metric Balancing

Achieving an optimal balance between FPR, query speed, and memory usage was difficult and required delicate tweaks. Each AMQ structure demonstrated unique trade-offs.

The Cuckoo Filter's dynamic resizing capabilities resulted in a trade-off with increased memory consumption, while the Vacuum Filter's advanced space management introduced implementation complexity.

5.2 Implementation Complexity

The reproduction of the results from the literate and the implementation of the various AMQ structures required strong algorithmic understanding. In particular, the Vacuum Filter presented difficult implementation challenges due to its advanced space management techniques. Debugging and ensuring the accuracy of these structures required rigorous testing and validation.

5.3 Scalability Limitations

Although the experiments showed promising results across various dataset sizes, the scalability of AMQ structures presented challenges. As dataset sizes increased from 10,000 items to 1,000,000 items, the time for performance analysis grew similarly, especially on machines with limited resources. The computational intensity of conducting the performance analysis for even larger datasets became apparent, as the analysis runs for the largest dataset (1,000,000 items) took several minutes to complete. In even larger datasets, tens of millions of records may exist, resulting in even longer computation time. This time complexity highlighted the need for more efficient approaches, and potentially better high-performing computational resources.

6 Conclusion

6.1 Summary

The study of Approximate Membership Query (AMQ) structures—Bloom Filters, Cuckoo Filters, and Vacuum Filters—has demonstrated their effectiveness in performing memory-efficient membership tests while enabling fast queries. Through an in-depth evaluation of these structures, this project explored both theoretical insights and practical applications. By extending AMQ structures to a new domain of basketball statistics, their versatility beyond conventional databases, networking, and caching systems was observed.

The basketball analytics application validated the scalability and performance of these filters under real-world data conditions, offering insights into how sports-related datasets can be managed more efficiently using modern data management techniques. The project also highlighted the trade-offs between different filters, emphasizing their suitability for specific tasks such as managing static data, handling dynamic updates, and supporting high-throughput querying.

6.2 Findings

6.2.1 Performance Comparison

For the FPR, the Vacuum Filter consistently maintained the lowest rate ($\tilde{0}.0002\%$), making it the most accurate filter for applications requiring membership tests. The Bloom Filter also performed well, with a FPR of ($\tilde{0}.01\%$), while the Cuckoo Filter lagged behind at $\tilde{0}.1\%$.

For the query speed, all three filters demonstrated similarly rapid speeds, which highlighted their negligible performance overhead. The Vacuum Filter maintained the highest query speed for the largest dataset and processed 1.16e-04 operations per second, making it ideal for real-time analytics. The Bloom Filter demonstrated fast performance due to its simple hashing mechanism, but was still outperformed. The Cuckoo Filter performed similarly, but was impacted by its bucket relocation process during insertion operations.

For memory usage, Bloom Filters proved to be the most memory-efficient, using only 1.2 MB for the largest dataset, followed closely by Vacuum Filters at 2.2 MB. The Cuckoo Filters required more memory (8.2 MB) due to the added complexity of maintaining buckets and fingerprints.

6.2.2 Implications for Basketball Analytics

The performance of the AMQ structures when applied to a dataset of basketball player statistics showed certain characteristics that highlight their potential in sports data management:

1. *Rapid Data Lookup*: Very fast query speeds enable quick access to player statistics, which is crucial for real-time analytics.
2. *Memory Efficiency*: Low memory usage allows for seamless integration into performance analysis tools and databases.
3. *Probabilistic Accuracy*: Minimal FPR ensures reliable data filtering and retrieval.

The analysis demonstrated that AMQ structures were well-suited for basketball analytics. The Bloom Filter emerged as particularly efficient for this domain due to its simplicity, and offered an optimal balance between FPR, query speed, and memory usage. Thus, sports analysts and data scientists can apply such probabilistic data structures to enhance real-time performance tracking, player comparison, and statistical analysis.

6.3 Future Work

In the future, the applications of AMQ structures in sports analytics can be extended and optimized through the following approaches:

- *Extended Dataset Testing*: Use larger datasets that include advanced basketball metrics like PER, true shooting percentage, and defensive ratings to evaluate performance with multi-dimensional statistics and larger datasets (for scalability testing).
- *Advanced Data Distributions*: Investigate how AMQs perform with non-uniform data distributions, applying techniques like range partitioning and load balancing.
- *Hybrid Filter Approaches*: Develop a hybrid structure combining the simplicity of Bloom Filters, the dynamic capabilities of Cuckoo Filters, and the efficiency of Vacuum Filters.
- *GPU-Accelerated Querying*: Explore GPU-based optimizations to improve query performance in real-time sports analytics platforms.
- *Integration into Sports Platforms*: Design and prototype a sports analytics platform that leverages AMQs for large-scale player data tracking, scouting, and game simulations.

7 Supporting Artifacts

Codebase: Repository includes Python scripts for AMQ implementations, performance analysis, and extensions to basketball analytics: Link
Dataset: 2023-24 NBA season's player statistics: Link

References

[1] Bin Fan, Dave G. Andersen, Michael Kaminsky, and Michael D. Mitzenmacher. Cuckoo filter: practically better than bloom. In *Proceedings of the 10th ACM international on conference on emerging networking experiments and technologies*, page 75–88. Association for Computing Machinery, 2014.

[2] Michael Mitzenmacher and Andrei Broder. Network applications of bloom filters: a survey. *Internet mathematics*, 1(4), 2004.

[3] Minmei Wang, Mingxun Zhou, Shouqian Shi, and Chen Qian. Vacuum filters: more space-efficient and faster replacement for bloom and cuckoo filters. *Proc. VLDB Endow.*, 13(2):197–210, 2019.