BionetBF: A Novel Bloom Filter for Faster Membership Identification of Paired Biological Network Data

Sabuzima Nayak[1*] and Ripon Patgiri[1]

[1*]Department of Computer Science & Engineering, National Institute of Technology Silchar, Silchar, 788010, Assam, India.

*Corresponding author(s). E-mail(s): sabuzima_rs@cse.nits.ac.in;
Contributing authors: ripon@cse.nits.ac.in;

Abstract

Biological network represents the interaction or relationship between the biological entities such as proteins and genes of a biological process. A biological network with thousands of millions of vertices makes its processing complex and challenging. In this article, we propose a novel Bloom Filter called Biological network Bloom Filter (BionetBF) fast membership identification of the biological network edges or paired biological data. BionetBF is capable of executing millions of operations within a second while occupying a tiny amount of main memory. We have conducted rigorous experiments to prove the performance of BionetBF with large datasets. The experiment is conducted using 12 generated datasets and three biological network datasets. It takes less than 8 sec for insertion and query of 40 million paired data. It demonstrates higher performance while maintaining a 0.001 false positive probability. BionetBF is also compared with other filters: Cuckoo Filter and Libbloom, where BionetBF proves its supremacy by exhibiting higher performance with a smaller sized memory compared with large sized filters of Cuckoo Filter and Libbloom.

Keywords: Bloom Filter/Biological Network/Genomic Data/Bioinformatics/Big Graph

1 Introduction

Biological network is a complex network that represents the relationship between biological entities [1]. A biological network consists of a set of vertices and edges where edges are the link that connects vertices based on some relationship. In other words, the vertices of the biological network are the same or different biological entities such as molecules, proteins, ions, metabolites, etc. Some examples of such biological network are protein-protein interaction networks [2], genetic regulatory networks, cell-cell communication [3], and genetic interaction networks [4]. Protein-protein interaction networks present transient and stable interactions between the proteins where the interactions are undirected by nature [5]. Stable interactions refer to the interaction within protein complexes, for instance, hemoglobin and ribosome; whereas transient interactions refer to the interactions occurring for a short duration to alter a protein which may alter further, for instance, protein kinases. Importantly, the undirected nature of the protein interaction elevates the complexity of identifying the involved proteins in the whole interaction. In the transcription process, the genome encodes many genes (usually in

the range of thousands) to produce various biological products such as proteins and mRNA [6]. The biological products are essential for various processes in the body, for instance, metabolism, cell survival, and cell differentiation. The gene regulatory network represents the complete set of gene products along with their interactions between them. Genetic interaction networks represent the interactions between the genes within an organism [7, 8]. Gene interactions transpire when a pair or more allelic or non-allelic genes of the same genotype interact to generate a specific phenotypic character.

Representing the biological system as a network instances increases the comprehension of the complex biological process, particularly at the molecular level [9]. Biological networks unfold prominent relationships, patterns, and properties of the whole system, which is difficult in a univariate analysis [10]. The study on biological network of protein-protein interaction helps in determining the protein related to the diseases [11, 12]. The interactions among the proteins result in cellular and molecular mechanisms that correspondingly determine an organism's health. Biological network is a great help in drug discovery. Typically, the commercial availability of a drug takes more than 7 years, the development of a drug takes approximately 5 years, and a few years more for clinical trials. Thus, the process of drug discovery is expensive and time consuming. Biological network helps in discovering the useful information related to a drug and its target, such as (a) determines any secondary effects interfering with the target, (b) whether the target protein has many connections, (c) whether the proteins of the targeted drug is dangerous, and (d) determines the molecules affected by the side effects of the drug. Furthermore, biological network is used for predicting the drug combination because the network is fitting for computational and combinatorial analysis. The study of genetic interaction networks helps in comprehending the relationship between phenotype and genotype. Phenotype is the trait of an organism that is resulted from the interaction of the genotype and its environment. The genotype is the collection of genes. Biological network also helps in determining how viruses and bacteria infect, persevere, and cause diseases [13, 14]. In addition, it helps to apprehend the pharmacokinetic and pharmacogenomic actions of antibacterial drugs. Furthermore, biological network helps in establishing the relationships among organisms, and species [15].

Analysis of large networks is predominantly computationally intractable [16]. In other words, the computationally intractable problems are either NP-hard or NP-complete problems that are solved approximately or heuristically. The various factors that contribute to the complexity of biological network are chemical kinetics, a vast number of interacting variables, feedback loops, dynamic in nature caused due to various linear or nonlinear relationships and stochasticity [17]. The biological network is dynamic in nature [18]; however, the information provided by the biological network is static at a instance. The biological network is highly influenced by external factors making it dynamic in nature. Comparatively, a biological network furnishes a small part of the whole biological network, hence, it is missing a lot of information. The biological network has thousands of millions of vertices with billions of interactions/edges. Often, the huge information the biological network generates yields dubious interaction. The construction of biological network is challenging due to the large biological data silo and the data are noisy [19]. Thus, it demands highly efficient data structures with low memory footprints.

Bloom Filter [20] is a membership checking probabilistic data structure. It's simple architecture helps in having a small memory footprint. Moreover, the operations have simple algorithms with constant time complexity. This acts as an advantage for deploying the Bloom Filter for filtering purposes in applications against Big data. Bloom Filter is extensively used in networking for packet filtering. For example, Bloom Filter is a very popular data structure against DoS and DDoS attacks [21]. Currently, Bloom Filter is explored in many other fields, for instance, Bioinformatics. One example of Bloom Filter application in Bioinformatics is k-mer counting. k-mer is a DNA substring of length k. The k-mer counting is a data reduction and error removal step. This step is expensive because in some assembling technique, it takes half of the total computation time. Some k-mer counting techniques based on Bloom Filter are Melsted and Pritchard [22], Turtle [23], Squeakr [24], and Mcvicar *et al.* [25].

Standard Bloom Filter/Bloom Filter is a bit array of m bits. It uses k number of hash functions. Initially, each bit/cell is set to 0. Bloom Filter performs three operations: insertion, query, and delete. In insertion operation, a single element (say biodata) is hashed by the k hash functions. The hash value gives a location of a cell. The k number of cells is set to 1. In query operation, the biodata is hashed by the k hash functions. If all the k cell locations are 1, then Bloom Filter returns true. If atleast one cell is 0, then Bloom Filter returns false. In delete operation, the k cell locations are obtained following the same procedure as insertion and query operation. The k cell locations are reset to 0. Bloom Filter has two main issues: false positive and false negative. False positive occurs when a biodata is not inserted into Bloom Filter, but during query, Bloom Filter returns true. Whereas false negative occurs when a biodata is inserted into Bloom Filter, but during query Bloom Filter returns false. Compared to false positives, false negatives are unacceptable because false positives can be detected later. But it is impossible to detect false negatives. Delete operation introduces false negatives, hence standard Bloom Filter does not permit delete operation. Another variant of Bloom Filter is introduced called counting Bloom Filter [26] to permit the delete operation. BionetBF is a variant of standard Bloom Filter. Hence, elaboration on counting Bloom Filter is avoided in the article.

Bloom Filter is a fast data structure for rapid membership identification while preserving a low memory footprint. It inspired us to propose a Bloom Filter for membership identification of paired biological data. In this article, a novel Bloom Filter, called BionetBF, is proposed for rapid membership identification of paired biological data of biological networks. BionetBF can perform insertion and query operations on millions of sequences within seconds. We have conducted extensive experiments on BionetBF using 12 generated biological datasets. BionetBF takes 7.6 sec for insertion of 40 million paired biological data. Similarly, it takes 7.6 sec for the query of 40 million paired biological data. BionetBF maintains a low false positive probability of 0.001. One important point to notice is that in many datasets BionetBF has zero false positive probability. The performance of BionetBF is measured using multiple parameters: Million operation per second (MOPS), Seconds per operation (SPO), and Megabyte per operation (MBPS). In insertion operation, the highest MOPS, SPO, and MBPS is 7.06, 1.87, and 162.78, respectively. Similarly, in query operation (Disjoint set), the highest MOPS, SPO, and MBPS is 6.95, 1.9, and 160.6, respectively. BionetBF demonstrates accuracy of more than 99.5%. BionetBF is compared with another two filters: Cuckoo Filter and Libbloom (standard Bloom Filter). A small sized BionetBF demonstrates faster operations and high performance than Cuckoo Filter and Libbloom. Cuckoo Filter requires 6.6× and Libbloom requires 4.7× larger memory than BionetBF. Furthermore, the performance of BionetBF is measured using three biological network datasets: Chemical-gene, Gene-Disease, and Gene-Gene. Chemical-gene is the dataset of interaction between genes and molecules. Gene-Disease is the dataset about the association of disease with genes. Gene-Gene is the dataset about the interaction among genes. BionetBF also presents higher performance in the biological network datasets. BionetBF can be used for quick identification of a specific paired biological data in many biological networks. BionetBF is also an excellent choice for filtering of repetitive insertion or query of paired biological data.

2 RESULTS

We have conducted extensive series of tests to validate the performance of BionetBF using diverse datasets: a 12 generated dataset (Section 4.4 provides the detail procedure of dataset construction) (Notation and file size details given in Table 5) and Biological network dataset (Downloaded from [27]) (File size given in Table 6). The experimental analysis shows the reduction of the memory footprint of BionetBF to construct a data structure for biological network data. We prove two major concerning points regarding biological network through conducting rigorous experiments: firstly, BionetBF performs fast processing of the overwhelming sized dataset using tiny amount of memory, and second is having lowest error. We have conducted the experiments in low-cost Ubuntu-Desktop computer with 4GB RAM and Core-i7 processor. BionetBF drastically reduces memory requirements, and we present its analysis and results in this section.

2.1 Selecting k for BionetBF

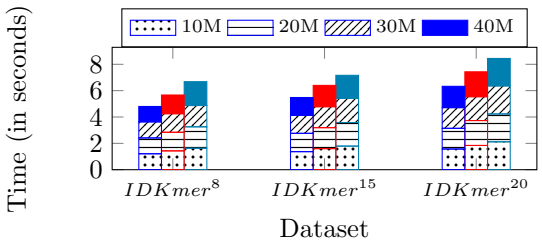
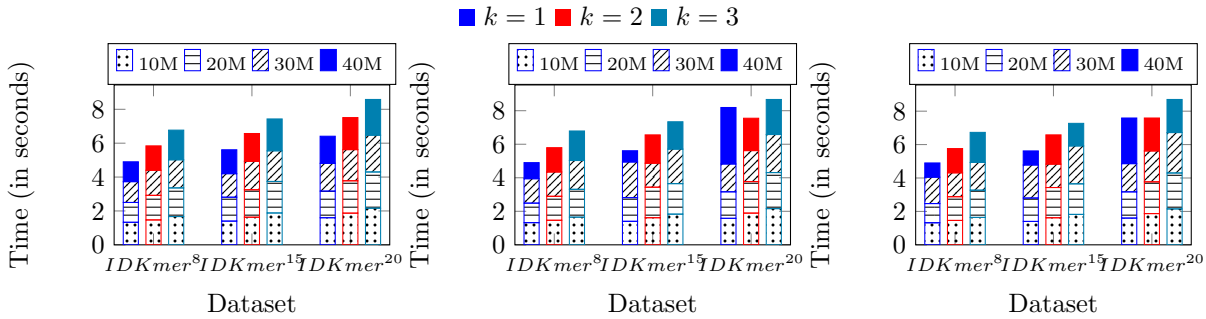


Fig. 1: Insertion Time (BionetBF k = 1 *vs* 2 *vs* 3): Comparison of insertion time among BionetBF having k = 1, 2 *and* 3. 10M: 10 million, 20M: 20 million, 30M: million, 40M: 40 million number of IDKmers inserted. Lower value is better.

In Bloom Filter, the number of hash functions influences the time complexity and FPP. Hence, first the number of hash functions (k) is determined for BionetBF to maintain the error within the desired FPP of 0.001. In this section, we have conducted experiments using the IDKmer dataset on BionetBF with k = 1 (say $BionetBF_{k=1}$), k = 2 (say $BionetBF_{k=2}$), and k = 3 (say $BionetBF_{k=3}$). Analysis of the figures is performed from the perspective of comparison among $BionetBF_{k=1}$, $BionetBF_{k=2}$ and $BionetBF_{k=3}$.

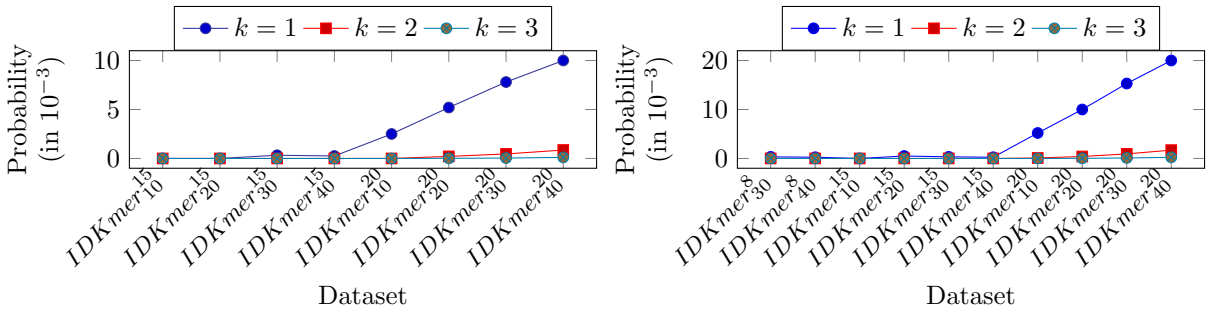
Figure 1 and Figure 2 illustrate the time taken by $BionetBF_{k=1}$, $BionetBF_{k=2}$ and $BionetBF_{k=3}$ for insertion and query operation respectively. In both figures, each bar is divided into four segments; (consider from bottom) the 1st, 2nd, 3rd, and 4th segment represents the 10 million, 20 million, 30 million and 40 million IDKmers. The time taken by a dataset is represented by the height of its corresponding segment from bottom. For example, time taken for 10 million IDKmers is the height of the 1st segment whereas time taken for 20 million IDKmers is the addition of height of 1st and 2nd segments. As shown in both figures, increase in the number of hash functions, increases the time duration of the operation. In insertion operation, $BionetBF_{k=1}$, $BionetBF_{k=2}$ and $BionetBF_{k=3}$ takes 6.3 sec, 7.4 sec, and 8.4 sec respectively for the highest sized IDKmer file i.e., $IDKmer^{20}_{40}$ (Size: 1.2GB). Whereas the least time taken for insertion is for $IDKmer^{8}_{10}$ dataset where $BionetBF_{k=1}$, $BionetBF_{k=2}$ and $BionetBF_{k=3}$ takes 1.21 sec, 1.43 sec, and 1.62 sec, respectively. Increase of the k value from 1 to 2, the insertion time increases by 0.59 sec on average and from 2 to 3 the insertion time increases by 0.57 sec on average. Figure 2a, Figure 2b, and Figure 2c exhibit the query time for Same Set, Mixed Set and Disjoint Set, respectively. In Figure 2a (Same Set), the minimum time taken by $BionetBF_{k=1}$, $BionetBF_{k=2}$ and $BionetBF_{k=3}$ is 1.33 sec, 1.47 sec, and 1.67 sec, respectively for $IDKmer^{8}_{10}$. The maximum time taken by $BionetBF_{k=1}$, $BionetBF_{k=2}$ and $BionetBF_{k=3}$ is 6.4 sec, 7.5 sec, and 8.59 sec, respectively for $IDKmer^{20}_{40}$. Increase of the k value from 1 to 2, the query time for Same Set increases by 0.61 sec on average and from 2 to 3 the query time increases by 0.59 sec on average. In Figure 2b (Mixed Set), the minimum time taken by $BionetBF_{k=1}$, $BionetBF_{k=2}$ and $BionetBF_{k=3}$ is 1.32 sec, 1.46 sec, and 1.65 sec, respectively for $IDKmer^{8}_{10}$. The maximum time taken by $BionetBF_{k=1}$, $BionetBF_{k=2}$ and $BionetBF_{k=3}$ is 8.19 sec, 7.56 sec, and 8.66 sec, respectively for $IDKmer^{20}_{40}$. Increase of the k value from 1 to 2, the query time for the Mixed Set increases by 0.39 sec on average and from 2 to 3 the query time increases by 0.6 sec on average. In Figure 2c (Disjoint Set), the minimum time taken by $BionetBF_{k=1}$, $BionetBF_{k=2}$ and $BionetBF_{k=3}$ is 1.32 sec, 1.46 sec, and 1.64 sec, respectively for $IDKmer^{8}_{10}$. The maximum time taken by $BionetBF_{k=1}$, $BionetBF_{k=2}$ and $BionetBF_{k=3}$ is 7.6 sec, 7.6 sec, and 8.7 sec, respectively for $IDKmer^{20}_{40}$. Increase of the k value from 1 to 2, the query time for Disjoint Set increases by 0.43 sec on average and from 2 to 3 the query time increases by 0.61 sec on average.

Figure 3 evince the comparison of $BionetBF_{k=1}$, $BionetBF_{k=2}$ and $BionetBF_{k=3}$ based on FPP. Figure 3a and Figure 3b represent the FPP for Mixed Set and Disjoint Set, respectively. BionetBF gives zero FPP in the $IDKmer^8$ with k = 1, 2, *and* 3. In case of Mixed Set (Figure 3a), $BionetBF_{k=1}$ gives more than desired FPP (i.e., 0.001) in $IDKmer^{20}$, on contrary, $BionetBF_{k=2}$ and $BionetBF_{k=3}$ gives FPP lower than 0.001 in all datasets. For $IDKmer^{20}_{40}$, $BionetBF_{k=1}$, $BionetBF_{k=2}$ and $BionetBF_{k=3}$ has FPP of 0.020, 0.0017, and 0.00023, respectively. In case of Disjoint Set (Figure 3b), $BionetBF_{k=1}$ gives more than 0.001 FPP in $IDKmer^{20}$ whereas $BionetBF_{k=3}$ has



(a) Same Set (b) Mixed Set (c) Disjoint Set

Fig. 2: **Query Time (BionetBF** $k = 1$ *vs* 2 *vs* 3**):** Comparison of time taken by BionetBF having $k = 1, 2$ *and* 3 for the Same Set, Mixed Set and Disjoint Set query operation. 10M: 10 million, 20M: 20 million, 30M: million, and 40M: 40 million number of IDKmer queries. Lower value is better.



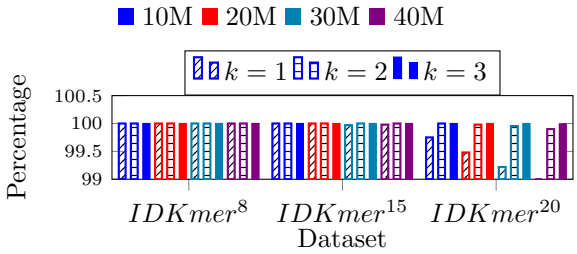
(a) Mixed Set (b) Disjoint Set

Fig. 3: **False Positive Probability (BionetBF** $k = 1$ *vs* 2 *vs* 3**):** Comparison of false positive probability of querying (a) Mixed Set and (b) Disjoint Set to BionetBF having $k = 1, 2$ *and* 3. Lower value is better. (a) IDKmer dataset have sequence length 15 and 20. BionetBF gives zero false positive probability for IDKmer dataset of sequence length 8. (b) BionetBF gives zero false positive probability in case of 10 and 20 million queries of IDKmer dataset of sequence length 8.

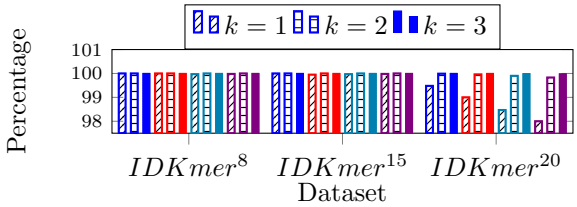
zero FPP in all datasets.$BionetBF_{k=2}$ gives 0.0017 FPP in $IDKmer^{20}_{40}$, but less than 0.001 FPP in other datasets. From the figures it is evident that $BionetBF_{k=1}$ has higher FPP than the desired FPP i.e., 0.001. $BionetBF_{k=3}$ has zero FPP, but takes more time for performing the operations. Thus, we have considered $k = 2$ for BionetBF based on the desired FPP and time duration of the operations.

BionetBF manifests 100% accuracy in all IDKmer dataset in Same Set because the BionetBF has zero false negative. The accuracy of Mixed Set and Disjoint Set is delineate in Figure 4a and Figure 4b, respectively, $BionetBF_{k=1}$, $BionetBF_{k=2}$ and $BionetBF_{k=3}$ exhibit 100% accuracy for $IDKmer^8$ and $IDKmer^{15}$. With $IDKmer^{20}_{40}$ dataset, $BionetBF_{k=1}$ demonstrates the lowest accuracy; however, it is 99% in the Mixed Set as shown in Figure 4a and 98% in the Disjoint Set as shown in Figure 4b.

Figure 5 highlights comparison among $BionetBF_{k=1}$, $BionetBF_{k=2}$ and $BionetBF_{k=3}$ based on the performance of insertion operation. Figure 5a demonstrates the comparison based on insertion MOPS. The highest MOPS of $BionetBF_{k=1}$, $BionetBF_{k=2}$ and $BionetBF_{k=3}$ is 8.31 ($IDKmer^8_{40}$), 7.06 ($IDKmer^8_{30}$), and 6.16 ($IDKmer^8_{30}$), respectively, whereas the lowest MOPS is 6.31 ($IDKmer^{20}_{40}$), 5.36 ($IDKmer^{20}_{20}$), and 4.72 ($IDKmer^{20}_{30}$), respectively. Figure 5b shows the comparison of performance based on insertion SPO. The highest SPO of $BionetBF_{k=1}$,

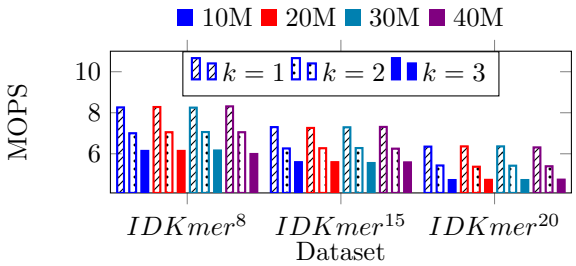


(a) Mixed set

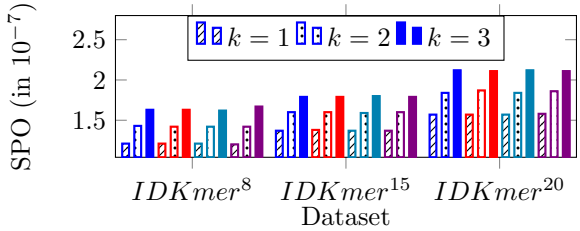


(b) Disjoint set

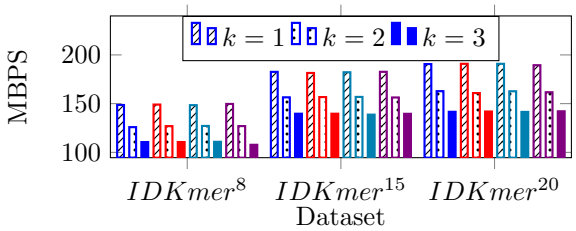
Fig. 4: Accuracy (BionetBF having $k = 1, 2$ *and* 3): Comparison of accuracy of querying Mixed Set and Disjoint Set to BionetBF having $k = 1, 2$ *and* 3. Accuracy of Same Set is 100%. Higher value is better. 10M: 10 million, 20M: 20 million, 30M: million, and 40M: 40 million number of IDKmer queries.



(a) Million operation per second (MOPS)



(b) Seconds per operation (SPO)



(c) Megabytes per second (MBPS)

Fig. 5: Performance of Insertion Operation (BionetBF having $k = 1, 2$ *and* 3): Comparison of performance among BionetBF having $k = 1, 2$ *and* 3 based on the (a) number of million insertion operations executed per second (MOPS), (b) time taken for execution of one insertion operation (SPO), and (c) execution of Megabyte per second in case of insertion operation (MBPS). In case of (a) and (c), higher is better. In case of (b), lower is better. 10M: 10 million, 20M: 20 million, 30M: million, and 40M: 40 million number of IDKmers inserted.

$BionetBF_{k=2}$ and $BionetBF_{k=3}$ is $1.58E-07$ ($IDKmer^{20}_{10}$), $1.87E-07$ ($IDKmer^{20}_{20}$), and $2.12E-07$ ($IDKmer^{20}_{30}$), respectively, while the lowest SPO is $1.2E-07$ ($IDKmer^{8}_{30}$), $1.42E-07$ ($IDKmer^{8}_{30}$), and $1.62E-07$ ($IDKmer^{8}_{30}$), respectively. Figure 5c exposes the differentiation among $BionetBF_{k=1}$, $BionetBF_{k=2}$ and $BionetBF_{k=3}$ based on MBPS for insertion operation. The highest MBPS exhibit by $BionetBF_{k=1}$, $BionetBF_{k=2}$ and $BionetBF_{k=3}$ is 190.9 ($IDKmer^{20}_{10}$), 162.78 ($IDKmer^{20}_{10}$), and 142.11 ($IDKmer^{20}_{40}$), respectively, and the lowest MBPS is 148.43 ($IDKmer^{8}_{30}$), 125.87 ($IDKmer^{8}_{10}$), and 107.77 ($IDKmer^{8}_{40}$), respectively. Figure 6 delineate the comparison among $BionetBF_{k=1}$, $BionetBF_{k=2}$ and $BionetBF_{k=3}$ based on the performance of query operation using MOPS. Figure 6a, Figure 6b and Figure 6c demonstrate on the query MOPS for Same Set, Mixed Set and Disjoint Set, respectively. In Same Set (Figure 6a), the highest MOPS is $BionetBF_{k=1}$, $BionetBF_{k=2}$ and $BionetBF_{k=3}$ is 8.16 ($IDKmer^{8}_{40}$), 6.86 ($IDKmer^{8}_{20}$), and 6 ($IDKmer^{8}_{10}$), respectively, whereas the lowest MOPS is 6.24 ($IDKmer^{20}_{30}$), 5.31 ($IDKmer^{20}_{20}$), and 4.62 ($IDKmer^{20}_{10}$), respectively. In Mixed Set (Figure 6b), $BionetBF_{k=1}$, $BionetBF_{k=2}$ and $BionetBF_{k=3}$ has the highest MOPS is 8.16 ($IDKmer^{8}_{40}$), 6.9 ($IDKmer^{8}_{20}$), and 6.05 ($IDKmer^{8}_{10}$), respectively, while the lowest MOPS of 4.88 ($IDKmer^{20}_{40}$), 5.29 ($IDKmer^{20}_{10}$), and 4.56 ($IDKmer^{20}_{30}$), respectively. In Disjoint Set (Figure 6c), $BionetBF_{k=1}$, $BionetBF_{k=2}$ and $BionetBF_{k=3}$ has the highest MOPS is

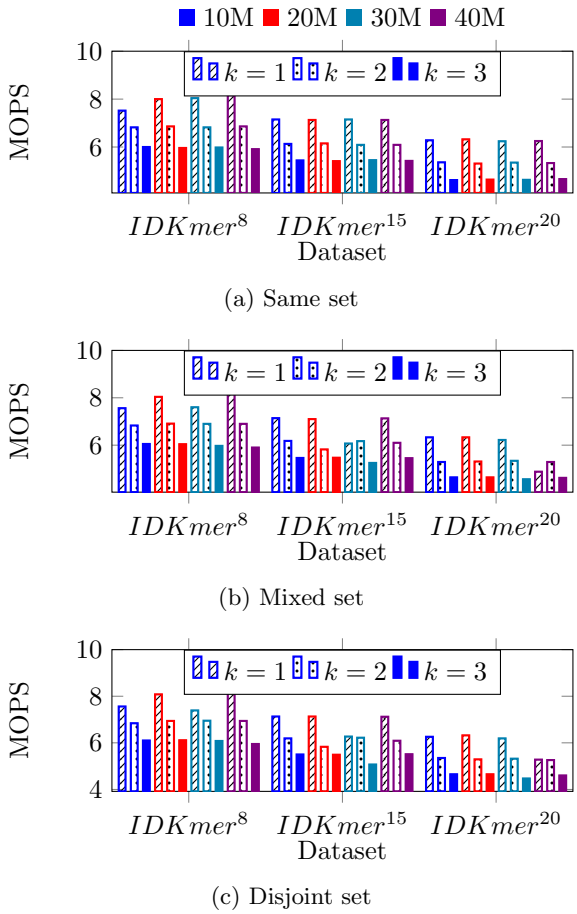


Fig. 6: **Query MOPS (BionetBF having $k = 1, 2$ *and* 3):** Comparison of performance among BionetBF having $k = 1, 2$ *and* 3 based on the million query operations executed per second (MOPS) for (a) Same set, (b) Mixed set, and (c) Disjoint set. Higher is better. 10M: 10 million, 20M: 20 million, 30M: million, and 40M: 40 million number of IDKmer queries.

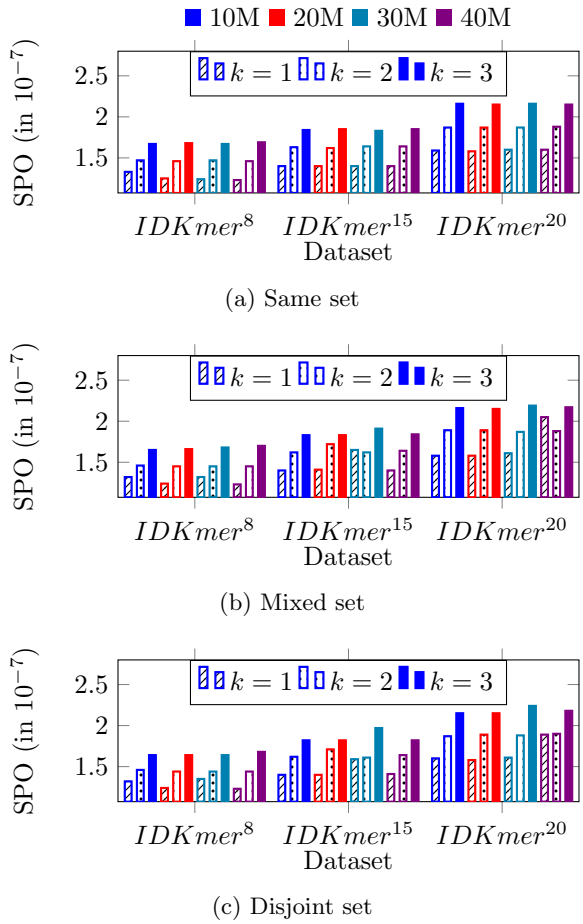


Fig. 7: **Query SPO (BionetBF having $k = 1, 2$ *and* 3):** Comparison of performance among BionetBF having $k = 1, 2$ *and* 3 based on second per operation (SPO) for (a) Same set, (b) Mixed set, and (c) Disjoint set. Lower is better. 10M: 10 million, 20M: 20 million, 30M: million, and 40M: 40 million number of IDKmer queries.

8.15 ($IDKmer^8_{40}$), 6.95 ($IDKmer^8_{30}$), and 6.11 ($IDKmer^8_{20}$), respectively, and the lowest MOPS of 5.28 ($IDKmer^{20}_{40}$), 5.26 ($IDKmer^{20}_{40}$), and 4.46 ($IDKmer^{20}_{30}$), respectively. Figure 7 shows the comparison of performance based on SPO for query operation. Figure 7a, Figure 7b and Figure 7c illustrate the query SPO for Same Set, Mixed Set and Disjoint Set, respectively. In Same Set (Figure 7a), the highest SPO in $BionetBF_{k=1}$, $BionetBF_{k=2}$ and $BionetBF_{k=3}$ is $1.6E-07$ ($IDKmer^{20}_{10}$), $1.89E-07$ ($IDKmer^{20}_{10}$), and $2.16E-07$ ($IDKmer^{20}_{10}$), respectively, whereas the lowest SPO is $1.23E-07$ ($IDKmer^8_{20}$), $1.46E-07$ ($IDKmer^8_{20}$), and $1.67E-07$ ($IDKmer^8_{10}$), respectively. In Mixed Set (Figure 7b), the highest SPO in $BionetBF_{k=1}$, $BionetBF_{k=2}$ and $BionetBF_{k=3}$ is $2.05E-07$ ($IDKmer^{20}_{10}$), $1.89E-07$ ($IDKmer^{20}_{10}$), and $2.19E-07$ ($IDKmer^{20}_{30}$), respectively, while the lowest SPO is $1.23E-07$ ($IDKmer^8_{30}$), $1.45E-07$ ($IDKmer^8_{20}$), and $1.65E-07$ ($IDKmer^8_{10}$), respectively. In Disjoint Set (Figure 7c), the highest SPO in $BionetBF_{k=1}$, $BionetBF_{k=2}$ and $BionetBF_{k=3}$ is $1.89E-07$ ($IDKmer^{20}_{10}$), $1.9E-07$ ($IDKmer^{20}_{40}$), and $2.24E-07$ ($IDKmer^{20}_{30}$), respectively, and the lowest

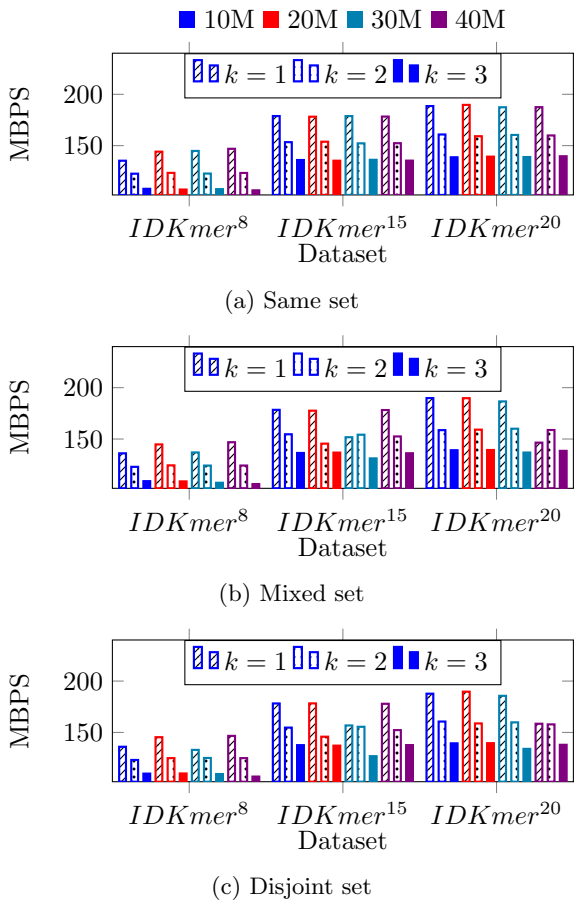


Fig. 8: **Query MBPS (BionetBF having $k = 1, 2$ *and* 3):** Comparison of performance among BionetBF having $k = 1, 2$ *and* 3 based on Megabyte per second (MBPS) for query operations in case of (a) Same set, (b) Mixed set, and (c) Disjoint set. Higher is better. 10M: 10 million, 20M: 20 million, 30M: million, and 40M: 40 million number of IDKmer queries.

SPO is $1.23E-07$ ($IDKmer^8_{30}$), $1.44E-07$ ($IDKmer^8_{30}$), and $1.654E-07$ ($IDKmer^8_{20}$), respectively. Figure 8 demonstrates the differentiation among $BionetBF_{k=1}$, $BionetBF_{k=2}$ and $BionetBF_{k=3}$ using performance based on MBPS for query operation. Figure 8a, Figure 8b and Figure 8c demonstrate the query in MBPS for Same Set, Mixed Set and Disjoint Set, respectively. Figure 8a demonstrates the query MBPS of the Same Set. The highest MBPS in $BionetBF_{k=1}$, $BionetBF_{k=2}$ and $BionetBF_{k=3}$ is 189.69 ($IDKmer^{20}_{10}$), 160.86 ($IDKmer^{20}_{10}$), and 139.66 ($IDKmer^{20}_{40}$), whereas, the lowest MBPS is 135.34 ($IDKmer^8_{20}$), 122.7 ($IDKmer^8_{10}$), and 106.46 ($IDKmer^8_{40}$), respectively. Figure 8b represents the query in MBPS of the Mixed Set. The highest MBPS in $BionetBF_{k=1}$, $BionetBF_{k=2}$ and $BionetBF_{k=3}$ is 189.87 ($IDKmer^{20}_{10}$), 160.06 ($IDKmer^{20}_{30}$), and 139.34 ($IDKmer^{20}_{20}$), while, the lowest MBPS is 136.05 ($IDKmer^8_{30}$), 122.95 ($IDKmer^8_{10}$), and 106.16 ($IDKmer^8_{40}$), respectively. Figure 8c represent the query MBPS of the Disjoint Set. The highest MBPS in $BionetBF_{k=1}$, $BionetBF_{k=2}$ and $BionetBF_{k=3}$ is 189.69 ($IDKmer^{20}_{10}$), 160.6 $IDKmer^{20}_{10}$), and 139.5 ($IDKmer^{20}_{20}$), and the lowest MBPS is 133 ($IDKmer^8_{30}$), 123.2 ($IDKmer^8_{10}$), and 107.02($IDKmer^8_{40}$), respectively.

We observe from Figure 5, Figure 6, Figure 7, and Figure 8 that the performance of $BionetBF_{k=3}$ is better than $BionetBF_{k=2}$ and $BionetBF_{k=1}$; $BionetBF_{k=2}$ is better than $BionetBF_{k=1}$. Considering the FPP, accuracy and performance, $BionetBF_{k=1}$ is less efficient. The $BionetBF_{k=3}$ is more efficient than $BionetBF_{k=2}$, but takes more time per operation. Thus, selecting $k = 2$ is the best choice for BionetBF.

2.1.1 Experimentation with higher k values:

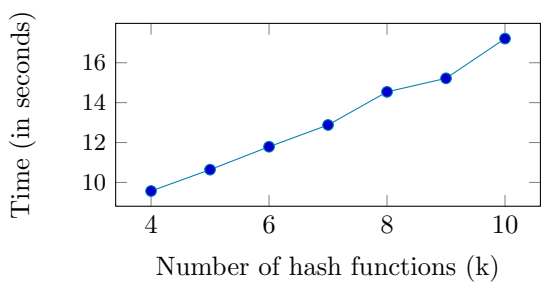


Fig. 9: **Insertion Time (BionetBF having $4 \leq k \leq 10$):** Comparison of insertion time among BionetBF having $4 \leq k \leq 10$. The dataset is IDKmer with sequence length 20 with 40 million IDKmers. Lower value is better.

We have further conducted experiments to illustrate the behaviour of BionetBF with higher k values, i.e., 4 to 10. All the experiment is performed on the $IDKmer^{20}_{40}$. Figure 9 presents the insertion time of BionetBF having $4 \leq k \leq$

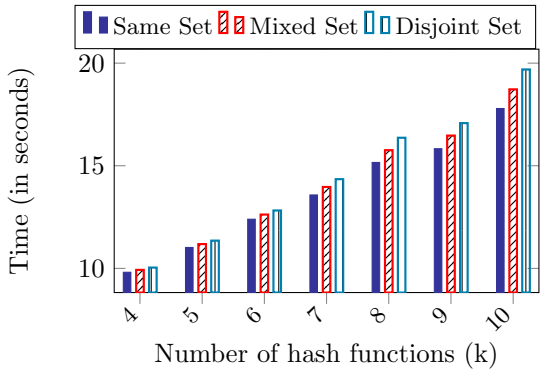


Fig. 10: **Query Time (BionetBF having $4 \leq k \leq 10$):** Comparison of query time among BionetBF having $4 \leq k \leq 10$. The dataset is IDKmer with sequence length 20 with 40 million queries. Lower value is better.

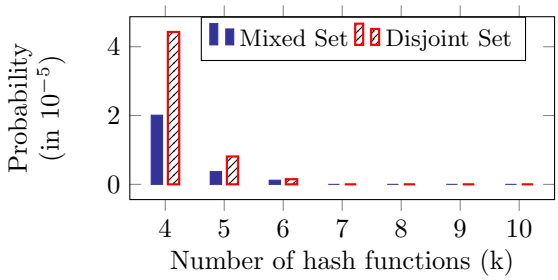


Fig. 11: **False Positive Probability (BionetBF having $4 \leq k \leq 10$):** Comparison of false positive probability of querying Mixed Set and Disjoint Set to BionetBF having $4 \leq k \leq 10$. The dataset is IDKmer with sequence length 20 with 40 million queries. Lower value is better.

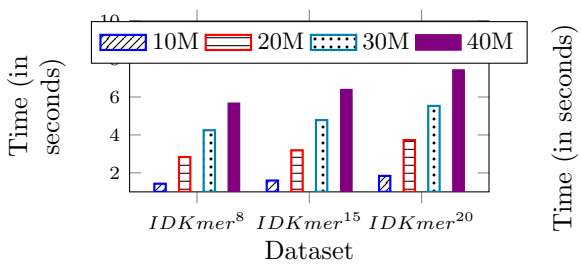
10. The $BionetBF_{k=4}$ takes 9.57 sec whereas $BionetBF_{k=10}$ takes 17.21 sec for insertion. An important point to notice is the time taken for insertion of the 1.2 GB IDKmer dataset while using 10 hash functions takes 17.21 sec, i.e. around one-fourth of a minute. On average, incrementing the k value increases the insertion time by 1.25 sec. Figure 10 depicts the query time of BionetBF having $4 \leq k \leq 10$. The $BionetBF_{k=10}$ takes 17.79 sec, 18.72 sec, and 19.69 sec for Same Set, Mixed Set and Disjoint Set, respectively. In other words, it takes less than a minute. On average, incrementing the k value increases the query time of Same Set, Mixed Set and Disjoint Set by 1.31 sec, 1.44 sec, and 1.57 sec, respectively. Figure 11 exposes the FPP of the BionetBF having $4 \leq k \leq 10$. BionetBF with $k = 6$ and higher, the FPP becomes zero for all datasets. The accuracy figure is not presented because the accuracy is 100% in all cases. From these experiments, one point can be highlighted that the k values of BionetBF can be increased to enhance the processing accuracy of the biological network datasets while maintaining less time duration for operation.

2.2 BionetBF

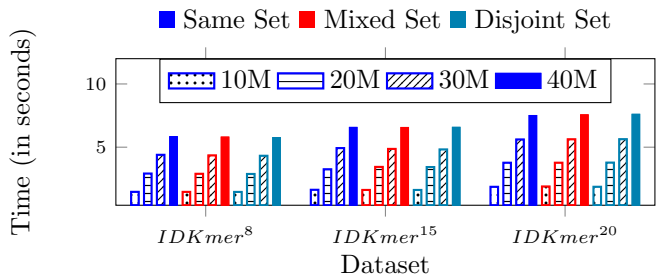
Number of IDKmers	BPS
10 million	10.85
20 million	5.425
30 million	3.612
40 million	2.71

Table 1: Bits Per Sequences (BPS) for BionetBF

This section presents the analysis on BionetBF with $k = 2$ using IDKmer dataset. From this section onwards the BionetBF refers to BionetBF with $k = 2$. Figure 12a represents the insertion time of BionetBF with different IDKmer datasets. BionetBF takes 1.43 sec for $IDKmer^{8}_{10}$ and 7.42 sec for $IDKmer^{20}_{40}$. On average, with every increment of 10 million IDKmer insertions, the time increases by 1.41 sec, 1.6 sec, and 1.86 sec for $IDKmer^{8}$, $IDKmer^{15}$, and $IDKmer^{20}$, respectively. The three average values vary even when having the same number of IDKmers because the sequence length is different. The IDKmer with a long sequence length requires more reading time. Figure 12b elucidate the time taken by BionetBF after querying the Same Set, Mixed Set and Disjoint Set of different IDKmer datasets. BionetBF for querying Same Set takes 1.47 sec for $IDKmer^{8}_{10}$ and 7.5 sec for $IDKmer^{20}_{40}$. In Same Set, with every increment of 10 million IDKmer queries, the time increases by 1.46 sec, 1.64 sec, and 1.88 sec for $IDKmer^{8}$, $IDKmer^{15}$, and $IDKmer^{20}$ dataset on average, respectively. BionetBF for querying the Mixed Set takes 1.46 sec for $IDKmer^{8}_{10}$ and 7.56 sec for $IDKmer^{20}_{40}$. In the Mixed Set, with every increment of 10 million IDKmer queries, the time increases by 1.44 sec, 1.65 sec, and 1.89 sec on average for $IDKmer^{8}$, $IDKmer^{15}$, and $IDKmer^{20}$, respectively. BionetBF for querying Disjoint Set takes 1.46 sec for $IDKmer^{8}_{10}$ and 7.6 sec for $IDKmer^{20}_{40}$.



(a) Insertion Time



(b) Query Time

Fig. 12: **Insertion and Query Time (BionetBF):** Comparison of time taken by BionetBF for (a) Insertion Operation and (b) Querying Operation. The Query operation is performed on Same Set, Mixed Set and Disjoint Set. Lower is better. 10M: 10 million, 20M: 20 million, 30M: million, and 40M: 40 million number of IDKmer queries.

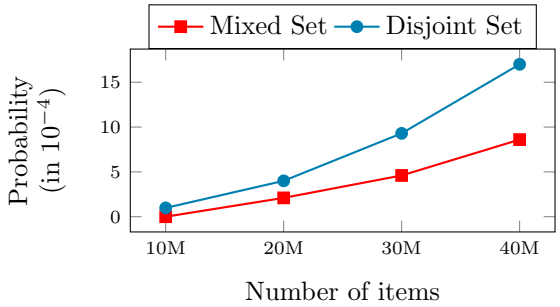
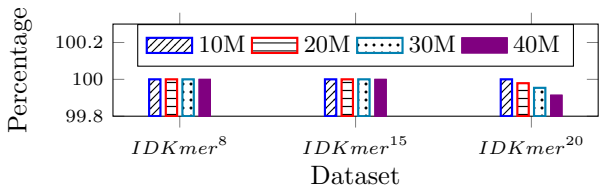
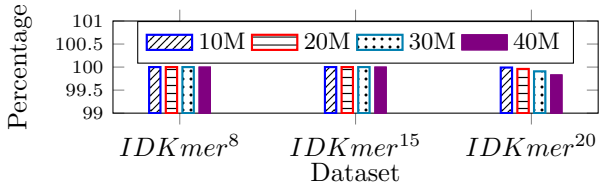


Fig. 13: **False Positive Probability (BionetBF):** Comparison of false positive probability of BionetBF; querying Mixed Set and Disjoint Set. The dataset is IDKmer having sequence length 20. The false positive probability of Mixed Set and Disjoint Set of IDKmer having sequence length 8 and 15 is zero. Lower is better. 10M: 10 million, 20M: 20 million, 30M: million, and 40M: 40 million number of IDKmer queries.



(a) Mixed Set

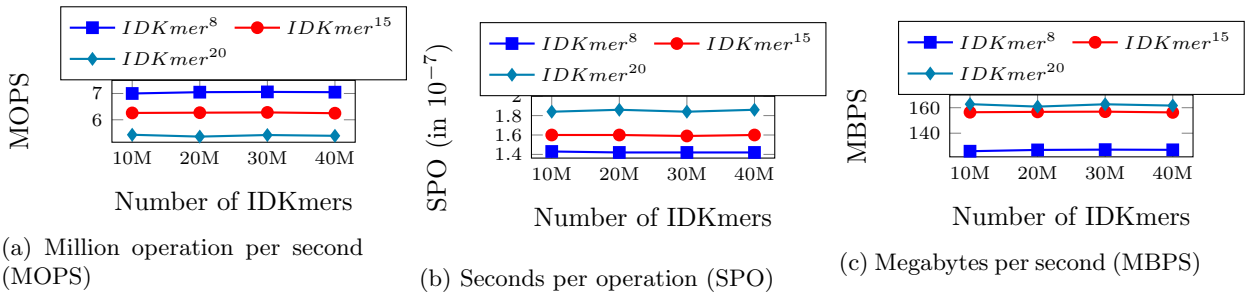


(b) Disjoint Set

Fig. 14: **Accuracy (BionetBF):** Comparison of accuracy of Same Set, Mixed Set and Disjoint Set queried to BionetBF. Accuracy of Same Set is 100%. Lower value is better. 10M: 10 million, 20M: 20 million, 30M: million, and 40M: 40 million number of IDKmer queries.

In the Disjoint Set, with every increment of 10 million IDKmer queries, the time increases by 1.43 sec, 1.65 sec, and 1.91 sec on average for $IDKmer^8$, $IDKmer^{15}$, and $IDKmer^{20}$, respectively. The insertion or query time of BionetBF is the combination of both reading and execution time. Thus, the operation time can be further increased if the whole dataset is kept in RAM; however, it is not possible for the biological network dataset because they are usually of very big size. Figure 13 highlights the FPP of the BionetBF. The Same Set gives the number of true positives equal to the number of IDKmers inserted. It indicates the FPP of BionetBF is zero. The $IDKmer^8$ and $IDKmer^{15}$ give zero FPP for both Mixed Set and Disjoint Set. The $IDKmer^{20}_{40}$ gives FPP of 0.00086 and 0.0017 for Mixed Set and Disjoint Set, respectively.

Figure 14 illuminates the accuracy of BionetBF which is 100% in the case of $IDKmer^8$ and $IDKmer^{15}$ for both Mixed Set and Disjoint Set. Moreover, the accuracy of the Same Set is 100%. Figure 14a and Figure 14b illustrate the accuracy of the Mixed Set and Disjoint Set, respectively. The accuracy of the Mixed Set and Disjoint Set for $IDKmer^{20}$ is 99.91% and 99.83%

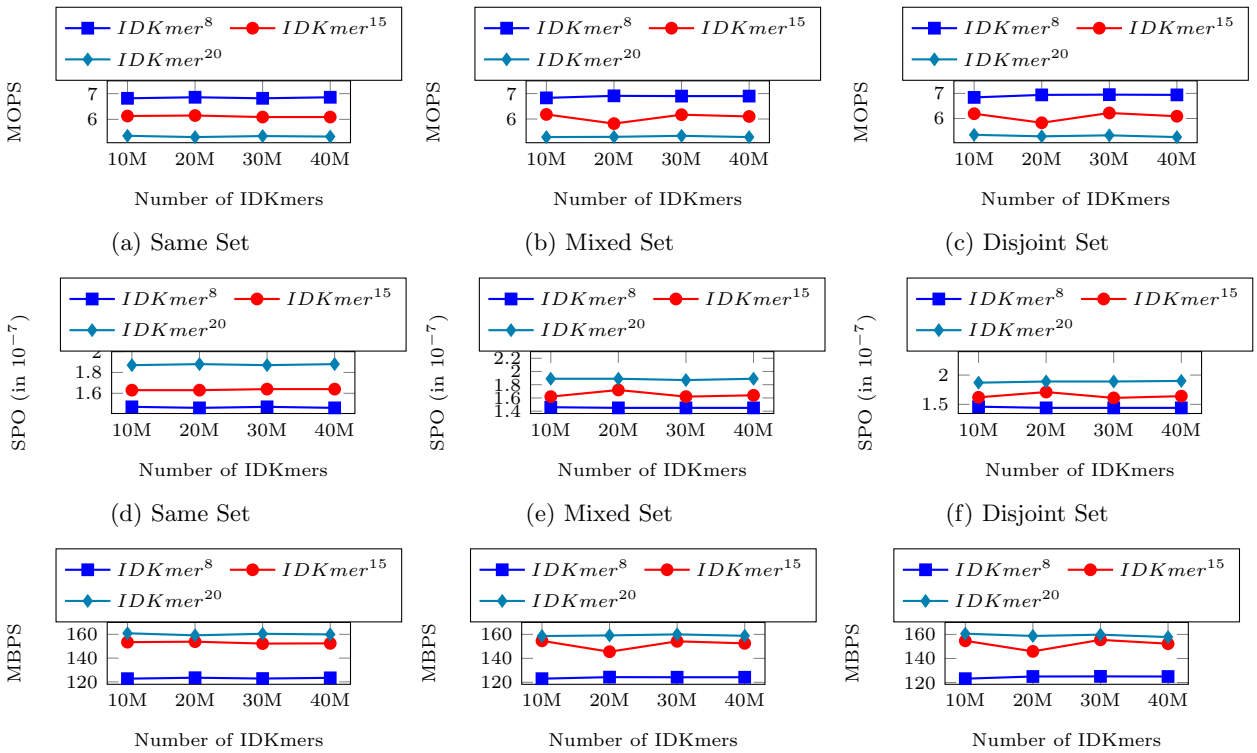


(a) Million operation per second (MOPS)

(b) Seconds per operation (SPO)

(c) Megabytes per second (MBPS)

Fig. 15: **Performance of Insertion Operation (BionetBF):** Comparison of performance of BionetBF based on the (a) number of million insertion operations executed per second (MOPS), (b) time taken for execution of one insertion operation (SPO), and (c) execution of Megabyte per second (MBPS) in case of insertion operation. In case of (a) and (c), higher is better. In case of (b), lower is better. 10M: 10 million, 20M: 20 million, 30M: million, and 40M: 40 million number of IDKmers inserted.



(a) Same Set

(b) Mixed Set

(c) Disjoint Set

(d) Same Set

(e) Mixed Set

(f) Disjoint Set

(g) Same Set

(h) Mixed Set

(i) Disjoint Set

Fig. 16: **Performance of Query Operation (BionetBF):** Comparison of performance of BionetBF based on [(a), (b) and (c)]: the execution of million query operations per sec (MOPS), [(d), (e) and (f)]: the time taken in seconds per operation (SPO), and [(g),(h) and (i)]: the Megabyte executed per second in case of query operation. The query operation is executed on Same Set, Mixed Set and Disjoint Set. In case of [(a), (b) and (c)] and [(g),(h) and (i)], higher is better. In case of [(d), (e) and (f)], lower is better. 10M: 10 million, 20M: 20 million, 30M: million, and 40M: 40 million number of IDKmer queries.

respectively. Thus, BionetBF has more than 99% accuracy for a large data file.

Figure 15 represents the performance of the insertion operation. Figure 15a exhibits MOPS

where the highest MOPS is 7.06 ($IDKmer^{8}_{30}$) and the lowest is 5.36 ($IDKmer^{20}_{20}$) in the IDKmer dataset. Figure 15b evince SPO in which the highest SPO is 1.87 ($IDKmer^{20}_{20}$) and the lowest is 1.42 ($IDKmer^{8}_{30}$). Figure 15c depicts MBPS where the highest MBPS is 162.78 ($IDKmer^{20}_{10}$) and the lowest is 125.87 ($IDKmer^{8}_{10}$). Similarly, Figure 16 delineates the performance of the query operation. Figure 16a, Figure 16b, and Figure 16c exhibit query MOPS for Same Set, Mixed Set and Disjoint Set, respectively. The highest MOPS for Same Set, Mixed Set and Disjoint Set is 6.86 ($IDKmer^{8}_{20}$), 6.9 ($IDKmer^{8}_{20}$), and 6.95 ($IDKmer^{8}_{30}$), respectively, whereas the lowest MOPS is 5.31 ($IDKmer^{20}_{20}$), 5.29 ($IDKmer^{20}_{10}$), and 5.26 ($IDKmer^{20}_{40}$), respectively. Likewise, Figure 16d, Figure 16e, and Figure 16f manifest query SOP for Same Set, Mixed Set and Disjoint Set, respectively. The highest SPO for Same Set, Mixed Set and Disjoint Set is $1.89E-07$ ($IDKmer^{20}_{20}$), $1.89E-07$ ($IDKmer^{20}_{10}$), and $1.9E-07$ ($IDKmer^{20}_{40}$), respectively, while the lowest SPO is $1.46E-07$ ($IDKmer^{8}_{20}$), $1.45E-07$ ($IDKmer^{8}_{20}$), and $1.44E-07$ ($IDKmer^{8}_{30}$), respectively. Furthermore, Figure 16g, Figure 16h, and Figure 16i show the query MBPS for Same Set, Mixed Set and Disjoint Set, respectively. The highest MBPS for Same Set, Mixed Set and Disjoint Set is 160.86 ($IDKmer^{20}_{20}$), 160.06 ($IDKmer^{20}_{30}$), and 160.6 ($IDKmer^{20}_{10}$), respectively, while the lowest MBPS is 122.7 ($IDKmer^{8}_{10}$), 122.95 ($IDKmer^{8}_{10}$), and 123.2 ($IDKmer^{8}_{10}$), respectively. Another parameter is used to determine the performance of BionetBF, i.e., the number of bits per sequence (BPS) (Equation (6)) illustrated in Table 1. The bits are calculated by the BionetBF data structure size. The BPS is based on two fixed parameters: BionetBF Bloom Filter size and the number of input sequences. Hence, BPS is the same for all IDKmer dataset with the same number of IDKmers. The highest BPS is 10.85 exhibited by the 10 million IDKmers, while the lowest is 2.71 exhibited by the 40 million IDKmers.

2.3 Comparison with Other Bloom Filters

In this section, BionetBF is compared with Cuckoo Filter [28] (Code available at [29]) and Libbloom (Code available at [30]). The filter size of the Cuckoo Filter is based on the total number of input sequences. In the experiments, the highest number of sequences inserted is 40 million; based on this, the filter size of Cuckoo Filter is 100MB. If the number of total sequences is reduced then, Cuckoo Filter does not insert all sequences in case of $IDKmer^{20}$. Cuckoo Filter takes numbers as input; hence, a hash function is added to convert the sequences to a number. Hence, the total number of hash functions used by the Cuckoo Filter is 3 if no kicking occurs. Another important point is if Cuckoo Filter is executed with the same dataset multiple times, it gives different false positives.

Filter	Filter Size	#Hash functions
Cuckoo Filter	100MB	3 (if no kicking)
Libbloom	71MB	10
BionetBF	15MB	2

Table 2: Details of Cuckoo Filter, Libbloom, and BionetBF. #Hash functions: Number of hash functions executed per operation.

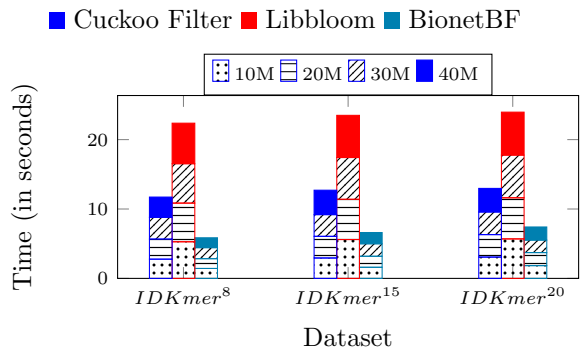
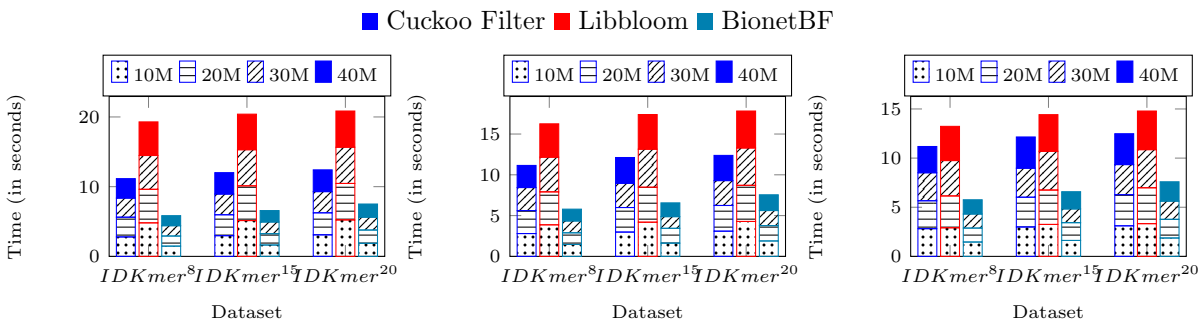


Fig. 17: **Insertion Time (Cuckoo Filter vs Libbloom vs BionetBF):** Comparison of insertion time among Cuckoo Filter, Libbloom, and BionetBF. 10M: 10 million, 20M: 20 million, 30M: 30 million, 40M: 40 million number of IDKmers inserted.

Number of IDKmers	Cuckoo Filter	Libbloom	BionetBF
10 million	80.53	57.51	10.85
20 million	40.27	28.76	5.43
30 million	26.84	19.17	3.61
40 million	20.13	14.38	2.71

Table 3: Comparison of memory usage among Cuckoo Filter, Libbloom, and BionetBF based on Bits Per Sequences (BPS).



(a) Same Set (b) Mixed Set (c) Disjoint Set

Fig. 18: **Query Time (Cuckoo Filter vs Libbloom vs BionetBF):** Comparison of time taken by Cuckoo Filter, Libbloom and BionetBF for the Same Set, Mixed Set and Disjoint Set for query operation. 10M: 10 million, 20M: 20 million, 30: million, and 40: 40 million number of IDKmers queried.

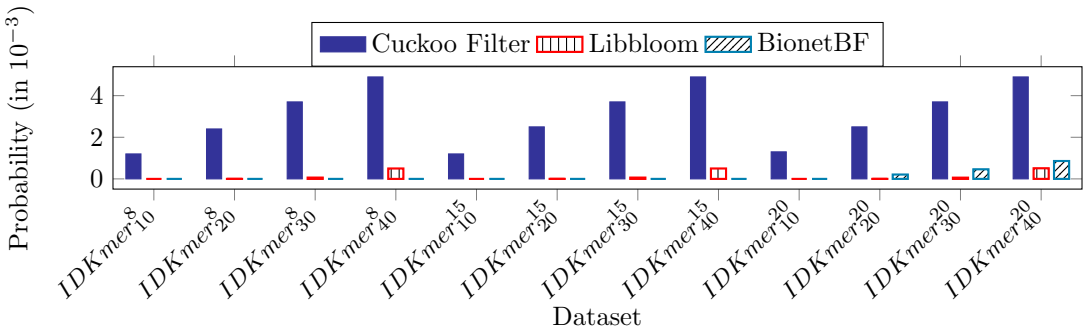


Fig. 19: **False Positive Probability (Libbloom vs Cuckoo Filter vs BionetBF):** Comparison of false positive probability of querying Mixed Set to Libbloom, Cuckoo Filter and BionetBF.

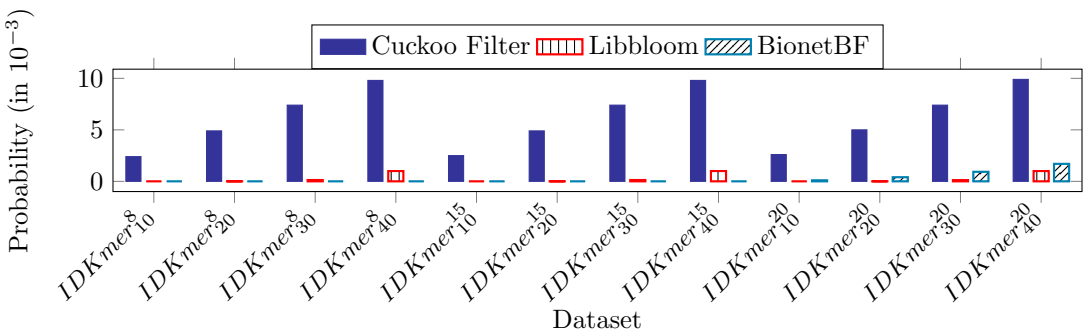
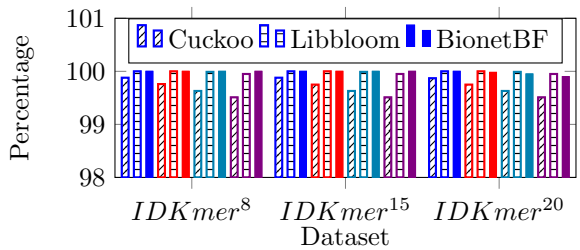
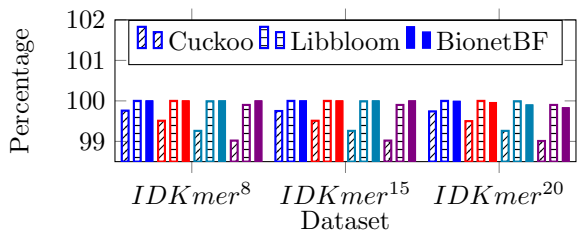


Fig. 20: **False Positive Probability (Libbloom vs Cuckoo Filter vs BionetBF):** Comparison of false positive probability of querying Disjoint Set to Libbloom, Cuckoo Filter and BionetBF.

We have experiment considered the least number of false positives after executing the Cuckoo Filter multiple times for a single dataset. The Libbloom is the standard Bloom Filter [20]. Similar to Cuckoo Filter, in the case of Libbloom, the total number of sequences considered is 40 million. The memory size is 71 MB and to achieve an FPP of 0.001, the Libbloom takes 10 hash functions.

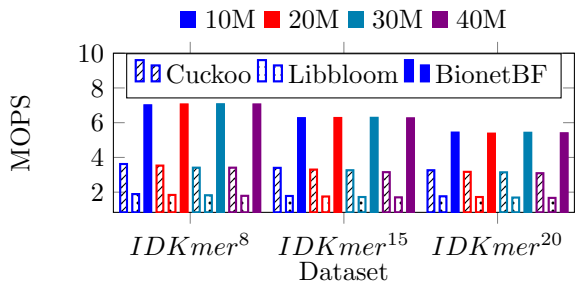


(a) Mixed set

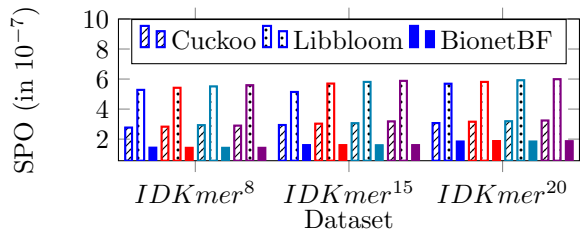


(b) Disjoint set

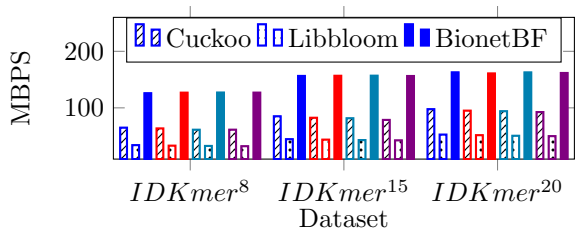
Fig. 21: **Accuracy (Cuckoo vs Libbloom vs BionetBF):** Comparison of accuracy of querying Same, Mixed and Disjoint set to Libbloom, Cuckoo, and BionetBF. Higher value is better. 10M: 10 million, 20M: 20 million, 30M: million, and 40M: 40 million number of IDKmers queried.



(a) Million operation per second (MOPS)



(b) Seconds per operation (SPO)



(c) Megabytes per second (MBPS)

Fig. 22: **Performance of Insertion Operation (Cuckoo vs Libbloom vs BionetBF):** Comparison of performance among Cuckoo, Libbloom, and BionetBF based on the (a) number of million insertion operations executed per second (MOPS), (b) time taken for execution of one insertion operation (SPO), and (c) execution of Megabyte per second (MBPS) in case of insertion operation. In case of (a) and (c), higher is better. In case of (b), lower is better. 10M: 10 million, 20M: 20 million, 30M: million, and 40M: 40 million number of IDKmers queried.

In the experiment, the filter size of BionetBF is 15MB. Similar to BionetBF, the Kmer ID and sequence of the IDKmer dataset are concatenated and inserted as a single sequence into Cuckoo Filter and Libbloom. Table 2 highlights the filter size and number of hash functions of Cuckoo Filter, Libbloom, and BionetBF, which are used in the experiment.

Figure 17 and Figure 18 show the comparison among Cuckoo Filter, Libbloom and BionetBF based on the time taken for insertion and query operation, respectively. Similar to Figure 1 and Figure 2, (considering from bottom) the 1st, 2nd, 3rd, and 4th segment of Figure 17 and Figure 18 represent the 10, 20, 30 and 40 million IDKmers, respectively. The Libbloom exhibits the highest time due to the highest number of hash functions as illuminated by Figure 17 and Figure 18. In the case of insertion of $IDKmer^{20}_{40}$, Cuckoo Filter, Libbloom and BionetBF take 12.97 sec, 23.95 sec, and 7.42 sec, respectively. Hence, Libbloom takes the longest time, and BionetBF takes the least time. With every increment of 10 million IDKmers, the insertion time of the Cuckoo Filter increases by on average 2.93, 3.18, and 3.24 sec for $IDKmer^8$, $IDKmer^{15}$, and $IDKmer^{20}$, respectively. In Libbloom, the insertion time increases by on average 5.59, 5.88 and 5.99 sec for $IDKmer^8$, $IDKmer^{15}$, and $IDKmer^{20}$, respectively. BionetBF takes 1.41, 1.6 and 1.86 sec more for $IDKmer^8$, $IDKmer^{15}$, and $IDKmer^{20}$, respectively with every increase in 10 million

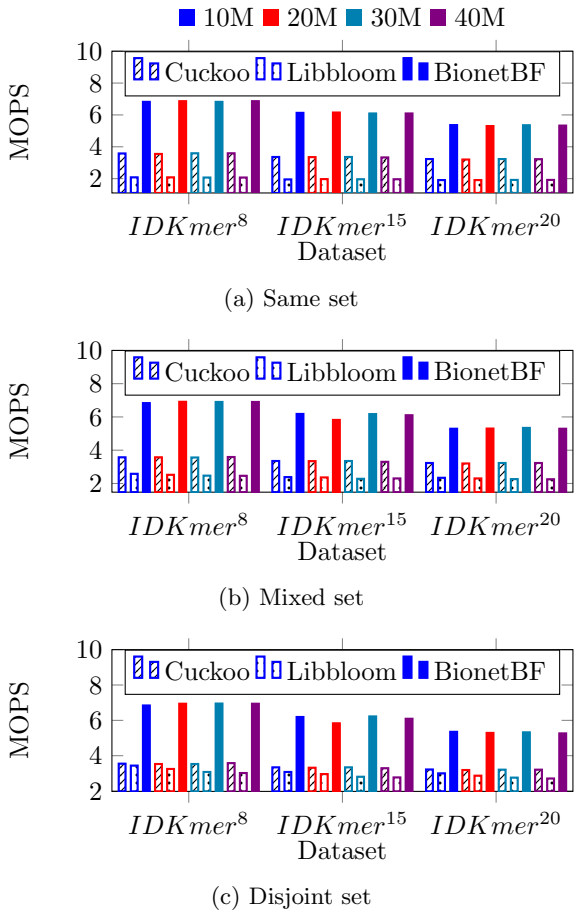


Fig. 23: **Query MOPS (Cuckoo vs Libbloom vs BionetBF):** Comparison of performance among Cuckoo, Libbloom, and BionetBF based on the million query operations executed per second (MOPS) for (a) Same set, (b) Mixed set, and (c) Disjoint set. Higher is better. 10M: 10 million, 20M: 20 million, 30M: million, and 40M: 40 million number of IDKmers queried.

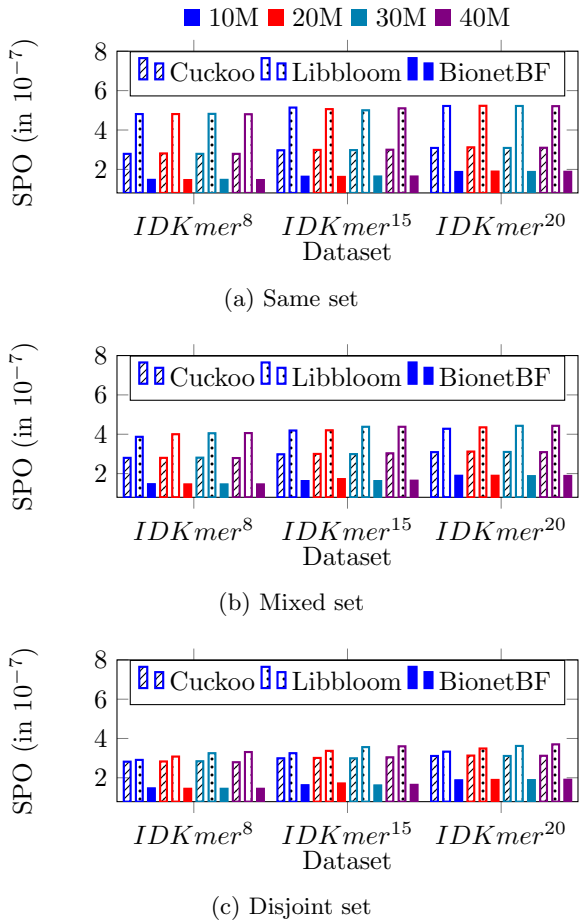


Fig. 24: **Query SPO (Cuckoo vs Libbloom vs BionetBF):** Comparison of performance among Cuckoo, Libbloom, and BionetBF based on second per operation (SPO) for (a) Same set, (b) Mixed set, and (c) Disjoint set. Lower is better. 10M: 10 million, 20M: 20 million, 30M: million, and 40M: 40 million number of IDKmers queried.

IDKmers. Figure 18a, Figure 18b, and Figure 18c elucidate the differentiation of Cuckoo Filter, Libbloom and BionetBF based on the query time of Same Set, Mixed Set, and Disjoint Set, respectively. In Same Set (Figure 18a), the query time taken for $IDKmer_{40}^{20}$ by Cuckoo Filter, Libbloom and BionetBF is 12.4 sec, 20.84 sec, 7.5 sec, respectively, whereas for $IDKmer_{10}^{8}$ is 2.79 sec, 4.81 sec, and 1.47 sec, respectively. Likewise, in Mixed Set (Figure 18b), the query time taken for $IDKmer_{40}^{20}$ by Cuckoo Filter, Libbloom and BionetBF is 12.38 sec, 17.82 sec, 7.56 sec, respectively; while for $IDKmer_{10}^{8}$ is 2.79 sec, 3.87 sec, and 1.46 sec, respectively. In Disjoint Set (Figure 18c), the query time taken for $IDKmer_{40}^{20}$ by Cuckoo Filter, Libbloom and BionetBF is 12.47 sec, 14.78 sec, 7.6 sec, respectively, and for $IDKmer_{10}^{8}$ is 2.82 sec, 2.91 sec, and 7.6 sec, respectively. Similar to insertion time, in case of query time, Libbloom takes the longest time, and BionetBF takes the least time. With every increment of 10 million IDKmer queries, the query time of Cuckoo Filter increases by on average 2.79, 3.04, and 3.12 sec for Disjoint Set of $IDKmer^8$, $IDKmer^{15}$, and $IDKmer^{20}$, respectively. In the case of Libbloom, the query time increases by on average 3.31, 3.6 and 3.7 sec

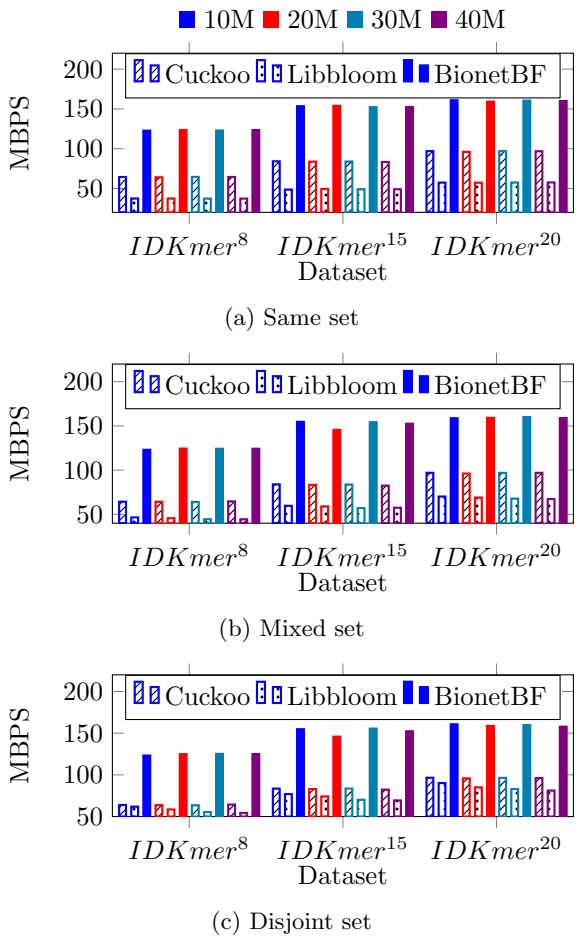


Fig. 25: **Query MBPS (Cuckoo vs Libbloom vs BionetBF):** Comparison of performance among Cuckoo filter, Libbloom, and BionetBF based on Megabyte per second for query operations in case of (a) Same set, (b) Mixed set, and (c) Disjoint set. Higher is better. 10M: 10 million, 20M: 20 million, 30M:million, and 40M: 40 million number of IDKmers queried.

for the Disjoint Set of $IDKmer^8$, $IDKmer^{15}$, and $IDKmer^{20}$, respectively. BionetBF takes 1.43, 1.65 and 1.91 sec more for the Disjoint Set of $IDKmer^8$, $IDKmer^{15}$, and $IDKmer^{20}$, respectively with every increment of 10 million IDKmer queries.

Figure 19 and Figure 20 delineate the comparison among Cuckoo Filter, Libbloom and BionetBF based on the FPP for Mixed Set and Disjoint Set. Clearly, the Cuckoo Filter has the highest FPP which is more than the desired FPP, i.e., 0.001 in all IDKmer datasets. Libbloom exhibits less than desired FPP in all IDKmer datasets. In the Mixed Set (Figure 19), Cuckoo Filter has 0.003 (approx.) on average more FPP than BionetBF and Cuckoo Filter has 0.0029 (approx.) on average more FPP than Libbloom. In the case of Libbloom, BionetBF has 0.0000144 (approx.) on average more FPP. It is an extremely low difference, but considering the size of the filters BionetBF has better performance. BionetBF has zero FPP in the majority IDKmer datasets, whereas except for the 10 million IDKmer datasets, Libbloom has some FPP. In the Disjoint Set (Figure20), the Cuckoo Filter has 0.0059 (approx.) on average more FPP than BionetBF, and the Cuckoo Filter has 0.006 (approx.) on average more FPP than Libbloom. In the case of Libbloom, it has 0.0000128 (approx.) on average less FPP than BionetBF. In the case of the Disjoint Set, BionetBF with its small filter size outperforms Libbloom. BionetBF has zero FPP in $IDKmer^8$ and $IDKmer^{15}$. Cuckoo Filter has zero FPP in all IDKmer datasets and Libbloom has zero FPP only in datasets having 10 million IDKmers. Therefore, BionetBF exhibits superior performance with regard to insertion time, query time and FPP with a small filter size for high number of IDKmers.

Figure 21 highlights the analogy among Cuckoo Filter, Libbloom and BionetBF based on the accuracy. In all IDKmer datasets, Libbloom and BionetBF have 100% accuracy, whereas Cuckoo Filter does not have 100% accuracy, but it is still more than 99%. This result is obvious because the data structure of Cuckoo Filter and Libbloom are constructed to achieve the desired FPP.

Figure 22 delineate comparison among Cuckoo Filter, Libbloom and BionetBF based on the performance of insertion operation. Figure 22a demonstrates the insertion MOPS of Cuckoo Filter, Libbloom and BionetBF. The highest insertion MOPS of Cuckoo Filter, Libbloom and BionetBF is 3.62 ($IDKmer^8_{10}$), 18.94 ($IDKmer^8_{10}$), and 7.06 ($IDKmer^8_{30}$), respectively, whereas the lowest MOPS is 3.09 ($IDKmer^{20}_{40}$), 1.67 ($IDKmer^{20}_{40}$), and 5.36 ($IDKmer^{20}_{20}$), respectively. BionetBF has the highest insertion MOPS with 1.88× and 3.53× (on average) better than Cuckoo Filter and Libbloom; the Cuckoo Filter is 1.88× (on average) better than Libbloom. Figure 22b illustrates the insertion SPO of Cuckoo Filter, Libbloom and BionetBF. The highest SPO of

Cuckoo Filter, Libbloom, and BionetBF is $3.24E-07$ ($IDKmer^{20}_{40}$), $5.99E-07$ ($IDKmer^{20}_{20}$), and $1.87E-07$ ($IDKmer^{20}_{20}$) respectively, while the lowest SPO is $2.77E-07$ ($IDKmer^{8}_{10}$), $5.28E-07$ ($IDKmer^{8}_{10}$), and $1.42E-07$ ($IDKmer^{8}_{20}$), respectively. In insertion SPO, Libbloom is highest, indicating bad performance compared to the other two; BionetBF is approximately 0.54× and 0.29× (on average) higher than Cuckoo Filter, and Libbloom and Cuckoo Filter is approximately 0.53 times (on average) better than Libbloom. Figure 22c highlights the insertion MBPS of Cuckoo Filter, Libbloom and BionetBF. The highest MBPS of Cuckoo Filter, Libbloom and BionetBF is 97.66 ($IDKmer^{20}_{10}$), 52.77 ($IDKmer^{20}_{10}$), and 162.78 ($IDKmer^{20}_{10}$), respectively, whereas the lowest MBPS is 61.36 ($IDKmer^{8}_{30}$), 32.19 ($IDKmer^{8}_{40}$), and 125.87 ($IDKmer^{8}_{10}$), respectively. In the case of MBPS, BionetBF is highest with approximately 1.88× and 3.53× (on average) better than Cuckoo Filter and Libbloom; the Cuckoo Filter is approximately 1.88× (on average) better than Libbloom. Similar to the performance of insertion operation, BionetBF has better performance in the query operation. Figure 23, Figure 24, and Figure 25 highlight the analogy among Cuckoo Filter, Libbloom and BionetBF based on the performance of query operation. Figure 23 represent the query MOPS of Cuckoo Filter, Libbloom and BionetBF where Figure 23a, Figure 23b, and Figure 23c highlights the query MOPS for Same Set, Mixed Set and Disjoint Set, respectively. In Same Set (Figure 23a), the highest MOPS shown by Cuckoo Filter, Libbloom and BionetBF is 3.59 ($IDKmer^{8}_{30}$), 2.08 ($IDKmer^{8}_{20}$), and 6.86 ($IDKmer^{8}_{20}$), respectively, whereas the lowest MOPS is 3.2 ($IDKmer^{20}_{20}$), 1.9 ($IDKmer^{20}_{20}$), and 5.31 ($IDKmer^{20}_{20}$), respectively. In Mixed Set (Figure 23b), the highest MOPS is Cuckoo Filter, Libbloom and BionetBF is 3.59 ($IDKmer^{8}_{40}$), 2.58 ($IDKmer^{8}_{10}$), and 6.9 ($IDKmer^{8}_{20}$), respectively, while the lowest MOPS is 3.2 ($IDKmer^{20}_{20}$), 2.25 ($IDKmer^{20}_{40}$), and 5.29 ($IDKmer^{20}_{10}$), respectively. In Disjoint Set (Figure 23c), the highest MOPS is Cuckoo Filter, Libbloom, and BionetBF is 3.58 ($IDKmer^{8}_{40}$), 3.44 ($IDKmer^{8}_{10}$), and 6.95 ($IDKmer^{8}_{30}$), respectively, and the lowest MOPS is 3.19 ($IDKmer^{20}_{20}$), 2.71 ($IDKmer^{20}_{40}$), and 5.26 ($IDKmer^{20}_{40}$), respectively. In MOPS, BionetBF is approximately 1.8×, 1.8× and 1.81× (on average) better than Cuckoo Filter in the Same Set, Mixed Set and Disjoint Set respectively, and approximately 3.07×, 2.56× and 2.048× (on average) better than Libbloom in Same Set, Mixed Set and Disjoint Set, respectively. The Cuckoo Filter is approximately 1.704×, 1.42×, and 1.13× (on average) better than Libbloom in Same Set, Mixed Set and Disjoint Set, respectively. Figure 24 shows the comparison of performance based on SPO for query operation. Figure 24a, Figure 24b and Figure 24c illustrate the query SPO for Same Set, Mixed Set and Disjoint Set, respectively. In Same Set (Figure 24a), the highest SPO in Cuckoo Filter, Libbloom and BionetBF is $3.12E-07$ ($IDKmer^{20}_{20}$), $5.23E-07$ ($IDKmer^{20}_{20}$), and $1.89E-07$ ($IDKmer^{20}_{20}$), respectively, whereas the lowest SPO is $2.79E-07$ ($IDKmer^{8}_{30}$), $4.81E-07$ ($IDKmer^{8}_{20}$), and $1.46E-07$ ($IDKmer^{8}_{20}$), respectively. In Mixed Set (Figure 24b), the highest SPO in Cuckoo Filter, Libbloom and BionetBF is $3.12E-07$ ($IDKmer^{20}_{20}$), $4.45E-07$ ($IDKmer^{20}_{40}$), and $1.89E-07$ ($IDKmer^{20}_{10}$), respectively, whereas the lowest SPO is $2.79E-07$ ($IDKmer^{8}_{40}$), $3.87E-07$ ($IDKmer^{8}_{10}$), and $1.45E-07$ ($IDKmer^{8}_{20}$), respectively. In Disjoint Set (Figure 24c), the highest SPO in Cuckoo Filter, Libbloom and BionetBF is $3.13E-07$ ($IDKmer^{20}_{20}$), $3.7E-07$ ($IDKmer^{20}_{40}$), and $1.9E-07$ ($IDKmer^{20}_{40}$), respectively, whereas the lowest SPO is $2.79E-07$ ($IDKmer^{8}_{40}$), $2.91E-07$ ($IDKmer^{8}_{10}$), and $1.44E-07$ ($IDKmer^{8}_{30}$), respectively. In SPO, BionetBF is approximately 0.5×, 0.56× and 0.55× (on average) better than Cuckoo Filter in the Same Set, Mixed Set and Disjoint Set, respectively, and approximately 0.33×, 0.39×, 0.49× (on average) better than Libbloom in Same Set, Mixed Set and Disjoint Set, respectively. The Cuckoo Filter is 0.59×, 0.7×, and 0.89× (on average) better than Libbloom in the Same Set, Mixed Set and Disjoint Set, respectively. Figure 25 demonstrates the differentiation among Cuckoo Filter, Libbloom, and BionetBF using performance based on query MBPS. Figure 25a, Figure 25b and Figure 25c illuminate the query MBPS for Same Set, Mixed Set and Disjoint Set, respectively. Figure 25a represents the query MBPS of Same Set where the highest MBPS in Cuckoo Filter, Libbloom, and BionetBF is 96.96 ($IDKmer^{20}_{10}$), 57.56 ($IDKmer^{20}_{40}$), and 160.86 ($IDKmer^{20}_{10}$), whereas the lowest MBPS is 63.99 ($IDKmer^{8}_{30}$), 37.32 ($IDKmer^{8}_{30}$), and 122.7

($IDKmer^8_{10}$), respectively. Figure 25b depicts the query MBPS of Mixed Set where the highest MBPS in Cuckoo Filter, Libbloom, and BionetBF is 96.96 ($IDKmer^{20}_{10}$), 70.13 ($IDKmer^{20}_{10}$), and 160.06 ($IDKmer^{20}_{30}$), while the lowest MBPS is 64 ($IDKmer^8_{10}$), 44.3 ($IDKmer^8_{40}$), and 122.95 ($IDKmer^8_{10}$), respectively. Figure 25c demonstrates the query MBPS of Disjoint Set where the highest MBPS in Cuckoo Filter, Libbloom, and BionetBF is 96.59 ($IDKmer^{20}_{10}$), 90.2 ($IDKmer^{20}_{10}$), and 160.6 ($IDKmer^{20}_{10}$), whereas the lowest MBPS is 63.46 ($IDKmer^8_{30}$), 54.41 ($IDKmer^8_{40}$), and 123.2 ($IDKmer^8_{10}$), respectively. In MBPS, BionetBF is approximately 1.8×, 1.8× and 1.81× (on average) better than Cuckoo Filter in the Same Set, Mixed Set and Disjoint Set, respectively, and approximately 3.07×, 2.56× and 2.048× (on average) better than Libbloom in Same Set, Mixed Set and Disjoint Set, respectively. The Cuckoo Filter is approximately 1.704×, 1.42×, and 1.13× (on average) better than Libbloom in Same Set, Mixed Set and Disjoint Set, respectively. Moreover, Table 3 highlights the comparison of performance among Cuckoo Filter, Libbloom, and BionetBF based on BPS. Cuckoo Filter occupies approximately 7.42× and 1.4× more BPS compared to Libbloom and BionetBF, respectively. Libbloom occupies approximately 5.3× more BPS than BionetBF.

From these experiments, we have proved the high efficiency and performance of BionetBF compared to other filters, namely, Cuckoo Filter and Libbloom. Another important point is that a small sized BionetBF has better performance than a big sized Cuckoo Filter and Libbloom. Another filter is also considered for comparison with BionetBF, namely, Xor filter [31] (Code available at [32]). For faster processing, the Xor filter saves all the sequences in an array which is given as input for the construction of the filter. This helps in reducing the time duration of the whole operation, i.e., insertion and query. Usually, genomic data is huge; inserting the whole genomic data in a single array is not possible. Moreover, the Xor filter has a fixed upper limit on the number of repetitive data possible in the dataset. The genomic data are highly repetitive in nature. This condition of the Xor filter is a huge constraint for the genomic data. Therefore, the XOR filter is not appropriate for genomic data.

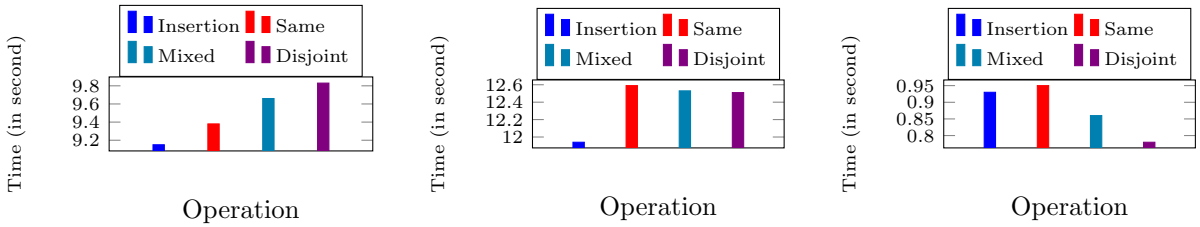
2.4 BionetBF with Biological network

Dataset	BPS
Chemical-Gene	1.73
Gene-Disease	1.34
Gene-Gene	22.51

Table 4: BPS of biological network

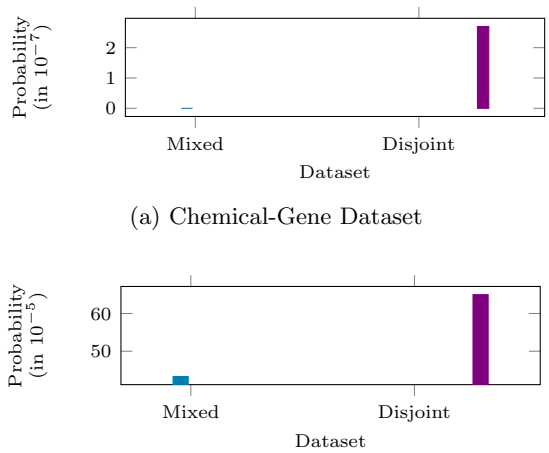
This section provides analysis and result of experimentation performed on BionetBF using three biological network datasets: Chemical-Gene, (b) Gene-Disease, and (c) Gene-Gene. Some details regarding the dataset are mentioned in Section Material and Methods (Data Description). While experimenting with the Chemical-Gene dataset, the paired biological data (i.e., chemical and gene) are concatenated and inserted or queried to BionetBF. Similarly, in the Gene-Disease dataset, the paired biological data (i.e., gene and chemical) are concatenated. In the case of Gene-Gene, the interacting genes are concatenated to insert or query to BionetBF. In this section, let the concatenated paired biological data is called bio sequences. An important point to highlight is the size of the Chemical-Gene dataset is 1.1 GB (approx.), Gene-Disease is 1.5GB (approx.), and Gene-Gene is 154MB.

Figure 26 represents the insertion and query time of BionetBF with the bionetwork dataset. Chemical-Gene, Gene-Disease, and Gene-Gene takes 9.15 sec, 11.937 sec and 0.925 sec respectively for insertion of bio sequences. Similarly, for Disjoint Set, the query time taken by Chemical-Gene, Gene-Disease, and Gene-Gene is 9.83 sec, 12.51 sec, and 0.775 sec respectively. Figure 27 illustrates the FPP of BionetBF with bionetwork datasets: Chemical-Gene (Figure 27a) and Gene-Gene (Figure 27b). Gene-Gene dataset has zero FPP. Both Chemical-Gene and Gene-Disease dataset are of big sized dataset, however, the FPP is very low. Figure 28 illuminates the accuracy of BionetBF with bionetwork datasets: Chemical-Gene (Figure 28a), Gene-Disease (Figure 28b) and Gene-Gene (Figure 28c). Both Chemical-Gene and Gene-Gene dataset have 100% accuracy; Gene-Disease has accuracy more than 99.9%. Figure 29 elucidate the performance of BionetBF with the bionetwork datasets for both insertion and



(a) Chemical-Gene Dataset (b) Gene-Disease Dataset (c) Gene-Gene Dataset

Fig. 26: **Insertion and Query Time (Biological network Dataset):** Time taken for Insertion and Query operation by BionetBF with biological network Dataset:- (a) Chemical-Gene, (b) Gene-Disease, and (c) Gene-Gene. Lower is better.



(a) Chemical-Gene Dataset

(b) Gene-Disease Dataset

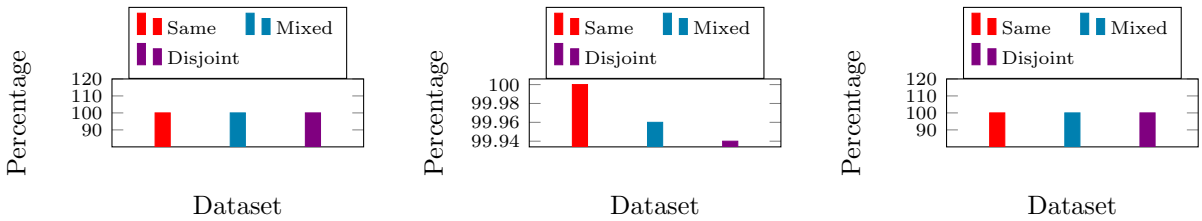
Fig. 27: **False Positive Probability (Biological network Dataset):** False positive probability of BionetBF with biological network dataset: (a) Chemical-Gene and (b) Gene-Disease. Lower is better. Gene-Gene dataset has zero false positive probability.

query operations. Considering both insertion and query operations, BionetBF with the bionetwork datasets exhibit more than 5 MOPS, below 2.10^{-7} SPO, and more than 110 MBPS.

3 Discussion

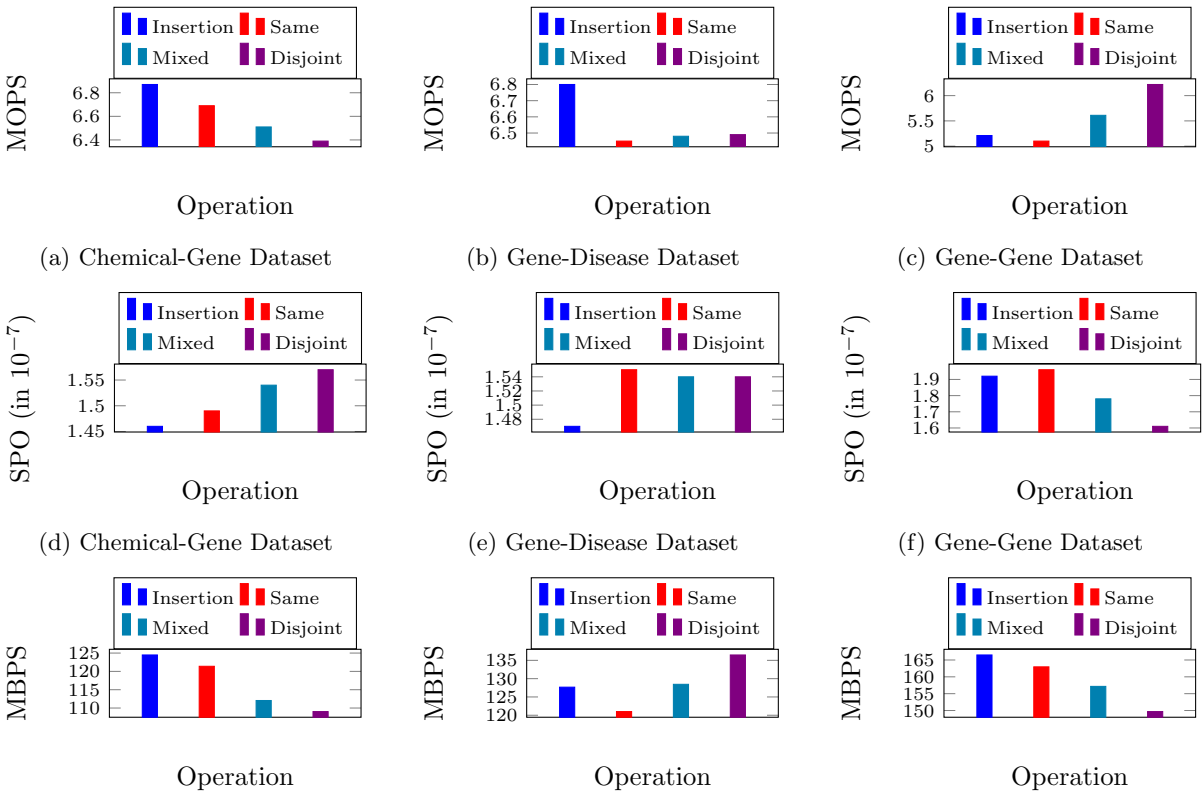
BionetBF can help in drug discovery. Another implementation is deduplication of the paired biological data. In addition, in this section some issues related to BionetBF are elaborated.

One implementation of BionetBF is the faster determination of the biological networks consisting of a particular target protein. The biological networks are stored in the database. Multiple BionetBFs are constructed for each biological network. Based on the size of the biological network, the size of the BionetBF is determined. A large biological network with thousands or millions of nodes can be stored in a small sized BionetBF. This is proved in the Result section. Hence, constructing a BionetBF is not much of an overhead. When a target protein and its other interacting protein is determined, this paired biological data is checked in BionetBFs parallelly. If a BionetBF returns true then its related biological network in the database is checked for further processing. Searching the biological network in the database is difficult because searching a big graph is an NP-hard problem. However, BionetBF can confirm the membership of a Big graph within seconds. Another application of BionetBF in drug discovery is security. The biological network can be stored in BionetBF and provided to others without disclosing the main structure of the biological network. Suppose a pharmaceutical company wants to study the compatibility of there drug with another drug belonging to another pharmaceutical company. The latter company does not want to completely provide the information about its drug (say, Drug A). However, they also want to encourage the development of the drug (say Drug B) by the former company. Drug B aims to determine whether consumption of Drug B along with Drug A is safe. Drug A company can store the protein-protein interactions in the BionetBF. Also, other information such as chemical-protein interaction can be stored in the same BionetBF. This BionetBF is provided to the Drug B company. BionetBF is itself an encrypted data; extracting complete information



(a) Chemical-Gene Dataset (b) Gene-Disease Dataset (c) Gene-Gene Dataset

Fig. 28: **Accuracy (Biological network Dataset):** Accuracy of BionetBF with biological network dataset:- (a) Chemical-Gene, (b) Gene-Disease, and (c) Gene-Gene. Higher is better.



(a) Chemical-Gene Dataset (b) Gene-Disease Dataset (c) Gene-Gene Dataset

(d) Chemical-Gene Dataset (e) Gene-Disease Dataset (f) Gene-Gene Dataset

(g) Chemical-Gene Dataset (h) Gene-Disease Dataset (i) Gene-Gene Dataset

Fig. 29: **Performance (Biological network dataset):** Performance of BionetBF with biological network dataset: [(a), (d), and (g)] Chemical-Gene, [(b), (e), and (h)] Gene-Disease, and [(c), (f), and (i)] Gene-Gene, based on [(a), (b), and (c)]: Million operation per second (MOPS). [(d), (e), and (f)]: Seconds per operation (SPO). [(g), (h), and (i)]: Megabyte executed per second (MBPS). In case of [(a), (b) and (c)] and [(g),(h) and (i)], higher is better. In case of [(d), (e) and (f)], lower is better.

from BionetBF is difficult. Even using a Brute force attack on the BionetBF, i.e., continuously querying the possible protein-protein interactions, is very difficult with such a huge number of protein possibilities. Drug B can determine its protein-protein interactions and check in the BionetBF whether it is present in Drug A.

Bloom Filters are mostly known for deduplication. BionetBF is also an excellent choice for

deduplication of repetitive insertion or query of paired biological data. While determining the biological network of a biological process, the same paired biological data is generated and forwarded to the database for storage. However, the repetition of such paired biological data can also be in thousands. Storage of huge repetitive information in databases is a huge issue in Bioinformatics. But, searching and removal of such repetitive biological data is another difficult task. Hence, in such a situation, BionetBF can be implemented. After generation of the paired biological data, it is checked in the BionetBF. In case it is present, the storage in the database is ignored; otherwise, it is stored in the BionetBF first then forwarded to the database for storage. Another advantage of using BionetBF is slow storage in the database without halting the processing. Databases are in secondary storage; hence, the storage time complexity is more. The application does not have to wait for the database for completion of the insertion of the biological data, rather BionetBF can continue working for the membership checking, and the newly paired biological data are stored in a buffer where it is slowly inserted into the database.

4 MATERIALS AND METHODS

4.1 Bloom Filter

Definition 1. *Bloom Filter is an approximation data structure to test whether a queried item is a member of a set or not. It returns either definitely not in the set or possibly in the set. An empty Bloom filter is a bit array of m bits, all set to* 0*. It requires k independent hash functions to map an input item into Bloom Filter. In insertion, all corresponding k bit positions are set to 1 using k hash functions. In a query operation, all k bit positions must be 1 to return true; otherwise, Bloom Filter returns false which refers that the queried item is not in the set [33].*

Bloom Filter [20] is an approximate set membership filtering data structure which is defined in Definition 1. It is extremely popular in diverse domains, particularly, Big Data [34], IoT, Cloud Computing, Networking [35], Security [21, 36], Database, Bioinformatics [37], and Biometrics. Bloom Filter is applied to reduce the main memory footprint. Bloom Filter uses a tiny amount of main memory to filter mammoth sized data. It can significantly boost up the performance of any system, but it has false positive probability issue. Nevertheless, the false positive probability can be mitigated by designing a new Bloom Filter.

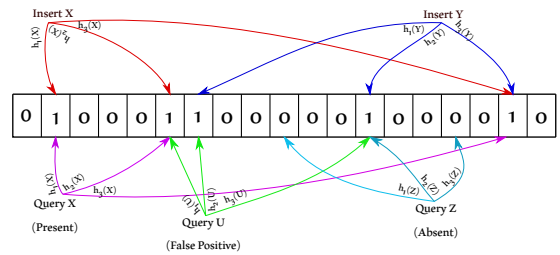


Fig. 30: Architecture of Standard Bloom Filter using three hash functions.

Bloom Filter [20] is a bitmap array consisting of bits. Let $\mathbb{B}$ be a Bloom Filter of size m, $\mathcal{S} = \{x_1, x_2, x_3, \ldots, x_n\}$ be set of words inserted into $\mathbb{B}$, U be the universe where $\mathcal{S} \subset$ U, and k is the number of hash functions. Bloom Filter size (i.e., m) is a prime number to reduce collision [33]. Bloom Filter calculates the optimal number of hash functions using Equation 1.

$$k = \frac{m}{n} ln2 \tag{1}$$

A large k value increases the time complexity of the operations, whereas a small k value increases collision probability. Bloom Filter returns either true or false in query operation, but these responses are further classified based on the presence/absence of the words in Bloom Filter. The *True* is classified into *True Positive* and *True Negative*, and *False* is classified into *False Positive* and *False Negative*. The definitions are as follows:

Definition 2. *If $x_1 \in \mathbb{S}$, and $x_1 \in \mathbb{B}$, then the result of $\mathbb{B}$ is true positive.*

Definition 3. *If $x_1 \notin \mathbb{S}$, but $x_1 \in \mathbb{B}$, then the result of $\mathbb{B}$ is false positive.*

Definition 4. *If $x_1 \notin \mathbb{S}$, and $x_1 \notin \mathbb{B}$, then the result of $\mathbb{B}$ is true negative.*

Definition 5. *If $x_1 \in \mathbb{S}$, but $x_1 \notin \mathbb{B}$, then the result of $\mathbb{B}$ is false negative.*

Figure 30 illustrates the bit array of a standard Bloom Filter. The Bloom Filter in the Figure 30 has $k = 3$, i.e., $h_1()$, $h_2()$, and $h_3()$, and X, Y, Z and Q are words. Initially the bit array of Bloom Filter is set to 0. The word X is inserted. The X is hashed by three hash functions. The hash value gives the bit location as pointed in Figure 30. Three hash functions give three bit locations which are set to 1. Similar procedure is followed to insert Y. In query operation, the word is hashed by the same hash functions to obtain the bit locations. The bit locations are checked whether all locations have the bit value as 1. If all bit locations are 1, then Bloom Filter returns true; if at least one bit location is 0, then Bloom Filter returns false. Query of X gives True response whereas query of Z gives False response. In case of query of Z, out of three bit locations, one bit location is 0; hence, Bloom Filter returns False.

Algorithm 1 Insertion of a word $Word$ into Bloom Filter $\mathbb{B}$ using three hash functions. The seed values $Seed$ S_1, S_2, S_3 are used to create different hash functions.

```
procedure INSERT(𝔹[], Word, S1, S2, S3)
    v1 = MURMUR(Word, length, S1)%m      ▷ MURMUR is Murmur hash function
    v2 = MURMUR(Word, length, S2)%m      ▷ m is the size of 𝔹[] which is a prime number.
    v3 = MURMUR(Word, length, S3)%m      ▷ % is modulus operator
    𝔹[v1] ← 1
    𝔹[v2] ← 1
    𝔹[v3] ← 1
end procedure
```

Bloom Filter has mainly three operations: insertion, query and deletion. Algorithm 1, 2, and 3 represent the insertion, query and deletion operations, respectively. As an example, the algorithms are written using the murmur hash function [38] with $k = 3$. Algorithm 1 inserts a word $Word$ into Bloom Filter $\mathbb{B}$. The modulus operation is performed on the hash value to limit the value within the size of the $\mathbb{B}$. Modulus operation gives three bit locations which are set to 1. Algorithm 2 query a word $Word$ into $\mathbb{B}$. The bit locations are obtained in the query operation by following the procedure similar to insertion operation. Then, the AND operations are performed to determine whether all bit locations are 1. If all bit locations are 1, then return True; otherwise false. Algorithm 3 deletes a word $Word$ from $\mathbb{B}$. Deletion operation also follows the same procedure to obtain the bit locations. Then, it performs AND operations to determine whether all the bit locations are 1. If all the bit locations are 1, then $Word$ is present in $\mathbb{B}$ and the bit locations are reset to 0.

Algorithm 2 Query a word $Word$ in Bloom Filter $\mathbb{B}[]$ using three hash functions. The seed values $Seed$ S_1, S_2, S_3 are used to create different hash functions.

```
procedure QUERY(𝔹[], Word, S1, S2, S3)
    v1 = MURMUR(Word, length, S1)%m
    v2 = MURMUR(Word, length, S2)%m
    v3 = MURMUR(Word, length, S3)%m
    if 𝔹[v1] AND 𝔹[v2] AND 𝔹[v3] then      ▷ AND-ing operation is perform to check the bit values of 𝔹[].
        return True
    else
        return False
    end if
end procedure
```

Algorithm 3 Deletion of a word $Word$ from Bloom Filter $\mathbb{B}[]$ using three hash functions. The seed values $Seed$ S_1, S_2, S_3 are used to create different hash functions.

```
procedure DELETE(𝔹[], Word, S1, S2, S3)
    v1 = MURMUR(Word, length, S1)%m
    v2 = MURMUR(Word, length, S2)%m
    v3 = MURMUR(Word, length, S3)%m
    if 𝔹[v1] AND 𝔹[v2] AND 𝔹[v3] then
        𝔹[v1] ← 0          ▷ Word is hashed into Bloom Filter array 𝔹[] to reset to 0
        𝔹[v2] ← 0
        𝔹[v3] ← 0
    else
        False
    end if
end procedure
```

There are two prominent issues in Bloom Filter, particularly false positive and false negative. Assuming the two words X and Y are inserted into $\mathbb{B}$. Consider Figure 30, a word Q is queried to $\mathbb{B}$, therefore, the correct response is False. When Q is queried, $\mathbb{B}$ hashes Q by three hash functions. The three hash functions obtained three locations. These locations are checked, all three locations have bit value 1. Hence, $\mathbb{B}$ returns True; which is an incorrect response. This happened because the locations obtained are already set to 1 by the words X and Y as illustrated in Figure 30. Another issue is false negatives, it happens due to deletion operations. Suppose the word X is deleted, then all the bits of these locations are reset to 0. But one location is common to both X and Y. Hence, after deletion of X, if Y is queried then $\mathbb{B}$ returns False. Even if Y is present in the Bloom Filter, it does not recognise the word. Thus, the delete operations are restricted in standard Bloom Filter. Therefore, Bloom Filter variants give more focus to reduce the false positive probability.

4.2 Related Work

There are many variants of Bloom Filter such as Counting Bloom Filter [26], rDBF [39], Stateful Bloom Filter [40], etc. Moreover, many filters are proposed for filtering, for example, Tinyset [41], Xor Filter [31], Cuckoo Filter [28], etc. To the best of our knowledge, no Bloom Filter or filters are proposed for biological network. We have chosen two filters to compare with our proposed algorithms: Cuckoo Filter and a standard Bloom Filter (Libbloom). BionetBF is compared with standard Bloom Filter, Libbloom and an approximate membership filter, Cuckoo Filter.

Cuckoo Filter [28] is a data structure used for membership inspection. Cuckoo Filter is not a Bloom Filter, rather it is a filter performing the same task as Bloom Filter, i.e., membership identification. It uses a modified cuckoo hashing [42], called partial-key Cuckoo hashing. This hashing improves dynamic insertion of words into the Cuckoo Filter. Instead of hash values, the fingerprint of the word is stored in the Cuckoo Filter to reduce memory requirement. The fingerprint size is less compared to the hashed value of the word. Two locations in the Cuckoo Filter are generated by hashing the fingerprint. Randomly any one location is selected for the storage of the fingerprint. In case the location is occupied, then the other location is selected. If both locations are occupied, then among the two locations, randomly any one location is selected. The fingerprint in the selected location is kicked, and the new fingerprint is stored in that location. The kicked fingerprint's alternate location is checked; if that location is empty, the kicked fingerprint is stored; otherwise, the kicking process is repeated. The kicking process continues until a threshold value. After crossing the kicking threshold, the Cuckoo Filter is confirmed as saturated. In query operation, the queried word is hashed to obtain the fingerprint. The fingerprint is again hashed to generate two locations; if it is present in any one location, then the Cuckoo Filter returns True; otherwise False.

4.3 BionetBF: The Proposed System

BionetBF is a novel Bloom Filter for membership identification of paired biological data. BionetBF is the modified 2DBF [39]. 2DBF is a fast Bloom Filter that uses fewer arithmetic operations. The BionetBF is a two-dimensional array where the dimensions are two different prime numbers. Considering prime numbers reduce collision. Usually, Bloom Filters take a single word as input, but the paired biological data has two data/words. Hence, the paired biological data are concatenated and given as input to BionetBF. This conversion of two words into a single word helps in maintaining a single data structure for biological dataset.

BionetBF implements two operations: insertion and query. The paired biological data are usually universal truths such as gene-gene interaction. Hence, there is no requirement for deletion of paired biological data from a Bloom Filter. Therefore, BionetBF does not implement delete operation. Let the paired biological data be (ID, $Kmer$) (say $IDKmer$) where ID is first biological data and $Kmer$ is second biological data. Kmer is a substring having length K within a biological sequence. Our desired FPP is 0.001. But, BionetBF with one hash function has more than 0.001 FPP. Hence, BionetBF uses two hash functions, this is experimentally illustrated in Results Section. Let, $\mathbb{B}_{x,y}$ is a BionetBF, (ID, $Kmer$) is a paired biological data, and $\mathcal{H}()$ is a hash function. BionetBF uses the murmur hash function [38]. In

the murmur hash function, different seed values create a different hash value for the same word. Initially, the BionetBF is set to 0.

Algorithm 4 demonstrates the insertion operation of BionetBF. For insertion operation, first, ID and $Kmer$ are concatenated (say the concatenated word as Sequence). Then the Sequence is hashed using the murmur hash function. After hashing, modular operations are performed on the hashed value using the dimensions of the BionetBF. The modular operations give a location in BionetBF which stores the information about the insertion of the Sequence into BionetBF. OR operation is performed to set one bit to 1 without losing previous information of that cell. This procedure is repeated twice as $k = 2$, in other words for each Sequence BionetBF sets two bits to 1.

Algorithm 4 Insertion of a paired biological data $(ID, Kmer)$ into BionetBF.

```
procedure IBionetBF(𝔹_{x,y}, ID, Kmer)   ▷ x and y are dimensions of BionetBF
    C_IDKmer = strcat(ID, Kmer)   ▷ Concatenating ID and Kmer
    for k ← 1 to 2 do
        if k == 1 then
            Seed ← Seed_1
        else
            Seed ← Seed_2
        end if
        h_IDKmer = H(C_IDKmer, Seed)   ▷ H() is murmur hash function
        i = h_IDKmer % x
        j = h_IDKmer % y
        pos = h_IDKmer % bits
        d = 𝔹_{i,j}
        p = 1 ≪ pos   ▷ ≪ is left shift operation
        𝔹_{i,j} = d OR p
    end for
end procedure
```

Algorithm 5 demonstrates the query operation on BionetBF. The steps till obtaining a location in BionetBF is the same for both insertion and query operation. The XOR operation removes the related information, i.e., it gives the bit value of the concerned Sequence. The AND operation determines whether the location bit is 1. When both location bits are 1 (as $k = 2$), then BionetBF returns True; otherwise False.

Algorithm 5 Query a paired biological data $(ID, Kmer)$ into BionetBF.

```
procedure QBionetBF(𝔹_{x,y}, ID, Kmer)
    C_IDKmer = strcat(ID, Kmer)   ▷ Concatenating ID and Kmer
    for k ← 1 to 2 do
        if k == 1 then
            Seed ← Seed_1
        else
            Seed ← Seed_2
        end if
        h_IDKmer = H(C_IDKmer, Seed)   ▷ H() is murmur hash function
        i = h_IDKmer % x
        j = h_IDKmer % y
        pos = h_IDKmer % bits
        d = 𝔹_{i,j}
        p = 1 ≫ pos   ▷ ≪ is right shift operation
        r = d ⊕ p
        t = r AND p
        if (t = 0) AND (d ≠ 0)) then
            flag ← 1
        else
            flag ← 0
        end if
    end for
    if flag == 1 then
        return true
    else
        return false
    end if
end procedure
```

4.4 Data Description

For conducting experiments on BionetBF, two datasets are considered: $IDKmer$ and biological network dataset. We preprocess the collected data to examine the performance of BionetBF. To determine the FPP and false negative probability, we have generated three different datasets for query operation: Same Set $\mathcal{S}$, Mixed Set $\mathcal{M}$ and Disjoint Set $\mathcal{D}$. Let, the inserted data set in the Bloom Filter be $\mathcal{I}$ (Inserted Set). The Same Set and Disjoint Set can be defined as $\mathcal{I} = \mathcal{S}$ and

$\mathcal{D} \cap \mathcal{S} = \phi$, respectively. Let, $\mathcal{M} = \{\mathcal{Q}_1,\ \mathcal{Q}_2\}$, then one of the following condition is true for Mixed Set:

1. If $\mathcal{Q}_1 \in \mathcal{I}$, then $\mathcal{Q}_2 \in \mathcal{D}$.
2. If $\mathcal{Q}_1 \in \mathcal{D}$, then $\mathcal{Q}_2 \in \mathcal{I}$.

To illustrate the above definitions,
Let $\mathcal{I} = \{(1, ATGA), (2, CGAG), (3, AGAG), (4, CAGC), (5, CATT),\ \ldots, (10, TTAG)\}$
Then, $\mathcal{S} = \{(1, ATGA), (2, CGAG), (3, AGAG), (4, CAGC), (5, CATT),\ \ldots, (10, TTAG)\}$
$\mathcal{D} = \{(11, TACT), (12, GCTC), (13, TCTC), (14, GTCG), (15, GTAA),\ \ldots, (20, AATC)\}$ and
$\mathcal{M} = \{(5, CATT), (6, ATTT), (7, TTTT),\ \ldots, (12, GCTC), (13, TCTC), (14, GTCG)\}$

The Same Set is not generated separately; rather, the $\mathcal{I}$ is again queried to BionetBF. To avoid confusion between insertion and query set, the Same Set reference is used to indicate the set which is queried. Disjoint Set is a set having no common paired biological data with the Inserted Set. The Mixed Set consists of half the number of paired biological data of Inserted Set and Disjoint Set. The Same Set is used to determine any false negative probability in BionetBF. Whereas, Mixed Set and Disjoint Set are used to determine FPP in BionetBF.

4.4.1 IDKmer Dataset:

A dataset called IDKmer was generated to experiment with BionetBF as a known dataset. The paired biological data of IDKmer are Kmer ID and Kmer. The second biological data of the paired biological data is a Kmer of a fixed length, and the first biological data is the ID of the Kmer. For the Kmer, a genomic sequence of a fixed length is read from a genomic dataset (Downloaded from [43]). The Kmer lengths considered are 8, 15 and 20. For assigning a unique Kmer ID, a 15 digit long number was incremented and assigned to each Kmer. The Disjoint Set is generated by taking a different 15 digit long number from the original set. There are four datasets with 10, 20, 30 and 40 million numbers of paired biological data having Kmer length 8. Similarly, IDKmer dataset with Kmer length 15 and 20, each has four datasets having 10, 20, 30 and 40 million numbers of paired biological data. Hence, there are 12 IDKmer datasets. Each 12 IDKmer datasets has its Mixed Set and Disjoint Set. Table 5 lists the notation used to refer to the datasets and the file size.

Dataset	**Denotation**	**File Size (in MB)**
$IDKmer^{X}_{Y}$	Kmer length is X with Y million IDKmers	NA
$IDKmer^{X}$	All IDKmer dataset of Kmer length X (i.e., 10, 20, 30 and 40 million IDKmers)	NA
$IDKmer^{8}_{10}$	Kmer length is 8 with 10 million IDKmers	180
$IDKmer^{8}_{20}$	Kmer length is 8 with 20 million IDKmers	360
$IDKmer^{8}_{30}$	Kmer length is 8 with 30 million IDKmers	540
$IDKmer^{8}_{40}$	Kmer length is 8 with 40 million IDKmers	720
$IDKmer^{15}_{10}$	Kmer length is 15 with 10 million IDKmers	250
$IDKmer^{15}_{20}$	Kmer length is 15 with 20 million IDKmers	500
$IDKmer^{15}_{30}$	Kmer length is 15 with 30 million IDKmers	750
$IDKmer^{15}_{40}$	Kmer length is 15 with 40 million IDKmers	1000
$IDKmer^{20}_{10}$	Kmer length is 20 with 10 million IDKmers	300
$IDKmer^{20}_{20}$	Kmer length is 20 with 20 million IDKmers	600
$IDKmer^{20}_{30}$	Kmer length is 20 with 30 million IDKmers	900
$IDKmer^{20}_{40}$	Kmer length is 20 with 40 million IDKmers	1200

Table 5: Notation and File size of $IDKmer$ Dataset. Mixed Set and Disjoint Set have the same file size. NA: Not Applicable.

4.4.2 Biological Network Dataset:

For the experiments, three biological network datasets are considered: Chemical-Gene, Gene-Disease, and Gene-Gene (Downloaded from [27]). In the Chemical-Gene dataset, the first biological data is chemical ID, and the second biological data is gene ID. The information of the dataset is the chemical-gene interaction network, i.e., the interaction between genes (proteins encoded by genes) and small molecules. In the Gene-Disease dataset, the first biological data is gene ID, and the second biological data is disease ID. The dataset provides the association of disease with genes. This

is known as disorder–gene association [44]. In the Gene-Gene dataset, both the biological data is gene ID. The interaction between gene and gene is also called Epistasis [45].

A Mixed Set and Disjoint Set is generated for each biological network dataset. The Disjoint Set is generated by incrementing the word "aaaaaaa" for Kmer ID and Kmer is the Gene sequence read from Chemical-Gene dataset. The Mixed Set is half of Chemical-Gene paired biological data from Chemical-Gene dataset and half paired biological data from the Disjoint Chemical-Gene dataset. Similarly, the Kmer ID of both Gene-Disease and Gene-Gene dataset is the incremented word starting from "aaaaaaa". The Kmer of the dataset is Disease and Gene sequence read for Gene-Disease and Gene-Gene dataset, respectively.

Dataset	File Size (in MB)
Chemical-Gene	1139
Mixed Chemical-Gene	1082
Disjoint Chemical-Gene	1072
Gene-Disease	1523
Mixed Gene-Disease	1610
Disjoint Gene-Disease	1707
Gene-Gene	154
Mixed Gene-Gene	135
Disjoint Gene-Gene	115

Table 6: File size of biological network Dataset.

4.5 Performance Parameters

Let $\mathcal{FPP}$, $\mathcal{FNP}$, $\mathcal{T}$, $\mathcal{N}$, $\mathcal{MB}$, $\mathcal{B}$, and n be the false positive probability, false negative probability, time taken (in seconds), number of operations (in million), file size in megabyte (MB), Bloom Filter's size (in bits), and number of input sequences, respectively. Equation (2), (3), (4), (5) and (6) define accuracy, million operations per seconds (MOPS), megabytes per second (MBPS), second per operation (SPO), and bits per item (BPS), respectively. These parameters are used to determine the performance of BionetBF.

$$Accuracy = 1 - (\mathcal{FPP} + \mathcal{FNP}) \quad (2)$$

$$Million\ Operations\ per\ Second\ (MOPS) = \frac{\mathcal{N}}{\mathcal{T}} \quad (3)$$

$$Second\ per\ Operations\ (SPO) = \frac{\mathcal{T}}{\mathcal{N}} \quad (4)$$

$$Megabytes\ Per\ Second\ (MBPS) = \frac{\mathcal{MB}}{\mathcal{T}} \quad (5)$$

$$Bits\ per\ Sequences\ (BPS) = \frac{\mathcal{B}}{n} \quad (6)$$

5 Conclusion

Our proposed system, BionetBF, identifies the membership of paired biological data effectively and efficiently. It can store a huge volume of paired biological data using a tiny amount of memory. As we know that biological data are significantly large, and it requires huge-sized memory (RAM) to process. Therefore, BionetBF offers an effective and efficient solution for the same. In addition, our proposed solution requires 2.71 bits per pair of biological data to store in the RAM. Therefore, it proves that it can store mammoth-sized biological data in a few MB main memory. BionetBF can process the biological data at the pace of 162.78 MBPS in a low-cost computer. Therefore, BionetBF will be able to process much higher MBPS in a higher-end computing system. We conducted a series of experiments using 12 generated biological datasets and three real biological network datasets to validate our proposed method. Experimentally, it is illustrated that the execution time of all the operations is low while mitigating the false positive probability to the lowest. Furthermore, BionetBF has a zero error rate in a few datasets. We compared BionetBF with another two filters: Cuckoo Filter and Libbloom (standard Bloom Filter), where a smaller sized BionetBF exhibits higher performance than the other two filters. Therefore, BionetBF can enhance the entire system performance of Drug Discovery, Genome coding, etc., due to its low memory footprint, nearly 100% accuracy, and high performance. Notably, BionetBF can dramatically boost up any system's performance. Our experimental results show the processing capabilities of biological data in low-cost hardware. Therefore, it can drastically reduce the computing cost of the entire application system.

BionetBF can be used for the storage of an extensive biological network with thousands or millions of nodes. It can be implemented for quick membership checking of neighboring nodes of a biological network for a specific protein. Moreover, it features a small memory footprint that can hold information about the huge biological network. Hence, an application can maintain multiple BionetBFs that stores nodes of biological networks for different proteins. In drug discovery, BionetBF can be used for passing information securely. BionetBF saves the biological network without disclosing the main structure. BionetBF is an outstanding choice for the deduplication of repetitive paired biological data. BionetBF offers constant time complexity, and it is ideal for the repetitive insertion or query of paired biological data. Thus, it can be a game-changer for genomics, proteomics, and Bioinformatics.

Acknowledgements

Author sincerely acknowledge the LabCrypt and BigCyber Researh Lab for providing the sufficient resource to carry out the entire research.

Ethics approval and consent to participate: Not applicable.

Authors' contributions:

Both authors have made equal contribution.

Availability of data and materials:

The source code is available at https://github.com/patgiri/BionetBF. The code is written in C programming language. All data are available at the given link. Moreover, we have also added a source code to generate synthesis data from a given data source.

Declaration

Consent for publication: Not applicable.

Competing interests: The authors declare that they have no competing interests.

Funding: None of the authors have received funding for this work.

References

[1] Badaloni, S., Camillo, B.D., Sambo, F.: Qualitative reasoning for biological network inference from systematic perturbation experiments. IEEE/ACM Transactions on Computational Biology and Bioinformatics **9**(5), 1482–1491 (2012). https://doi.org/10.1109/TCBB.2012.69

[2] Taipale, M.: Two protein/protein interaction assays in one go. Molecular systems biology **14**(7), 8485 (2018)

[3] Shao, X., Lu, X., Liao, J., Chen, H., Fan, X.: New avenues for systematically inferring cell-cell communication: through single-cell transcriptomics data. Protein & cell **11**, 866–880 (2020)

[4] Kwoh, C., Ng, P.: Network analysis approach for biology. Cellular and Molecular Life Sciences **64**(14), 1739–51 (2007)

[5] Singh, A.J., Ramsey, S.A., Filtz, T.M., Kioussi, C.: Differential gene regulatory networks in development and disease. Cellular and Molecular Life Sciences **75**(6), 1013–1025 (2018)

[6] Zhao, L.-Y., Song, J., Liu, Y., Song, C.-X., Yi, C.: Mapping the epigenetic modifications of dna and rna. Protein & Cell **11**, 792–808 (2020)

[7] Walhout, A.J.: What does biologically meaningful mean? a perspective on gene regulatory network validation. Genome biology **12**(4), 1–7 (2011)

[8] Boucher, B., Jenna, S.: Genetic interaction networks: better understand to better predict. Frontiers in Genetics **4**, 290 (2013). https://doi.org/10.3389/fgene.2013.00290

[9] Lecca, P., Priami, C.: Biological network inference for drug discovery. Drug Discovery Today **18**(5), 256–264 (2013). https://doi.org/10.1016/j.drudis.2012.11.001

[10] Kohn, K.W., Aladjem, M.I.: Circuit diagrams for biological networks. Molecular systems biology **2**(1), 2006–0002 (2006)

[11] Kuzmanov, U., Emili, A.: Protein-protein interaction networks: probing disease mechanisms using model systems. Genome medicine **5**(4), 1–12 (2013)

[12] Wetie, A.G.N., Sokolowska, I., Woods, A.G., Roy, U., Deinhardt, K., Darie, C.C.: Protein–protein interactions: switch from classical methods to proteomics and bioinformatics-based approaches. Cellular and molecular life sciences **71**(2), 205–228 (2014)

[13] Wang, E., Lenferink, A., O'Connor-McCourt, M.: Cancer systems biology: exploring cancer-associated genes on cellular networks. arXiv preprint arXiv:0712.3753 (2007)

[14] Mulder, N.J., Akinola, R.O., Mazandu, G.K., Rapanoel, H.: Using biological networks to improve our understanding of infectious diseases. Computational and Structural

Biotechnology Journal **11**(18), 1–10 (2014). https://doi.org/10.1016/j.csbj.2014.08.006

[15] Ogata, H., Fujibuchi, W., Goto, S., Kanehisa, M.: A heuristic graph comparison algorithm and its application to detect functionally related enzyme clusters. Nucleic acids research **28**(20), 4021–4028 (2000)

[16] Moreno, L.Q.: Graphlets and motifs in biological networks. In: Ranganathan, S., Gribskov, M., Nakai, K., SchÃ¶nbach, C. (eds.) Encyclopedia of Bioinformatics and Computational Biology, pp. 814–820. Academic Press, Oxford (2019). https://doi.org/10.1016/B978-0-12-809633-8.20291-4

[17] Zhang, B., Tian, Y., Zhang, Z.: Network biology in medicine and beyond. Circulation: Cardiovascular Genetics **7**(4), 536–547 (2014)

[18] Ideker, T., Krogan, N.J.: Differential network biology. Molecular systems biology **8**(1), 565 (2012)

[19] Myers, C.L., Robson, D., Wible, A., Hibbs, M.A., Chiriac, C., Theesfeld, C.L., Dolinski, K., Troyanskaya, O.G.: Discovery of biological networks from diverse functional genomic data. Genome biology **6**(13), 1–16 (2005)

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

[21] Patgiri, R., Nayak, S., Borgohain, S.: Preventing ddos using bloom filter: A survey. ICST Transactions on Scalable Information Systems **5**(19), 155865 (2018). https://doi.org/10.4108/eai.19-6-2018.155865

[22] Melsted, P., Pritchard, J.K.: Efficient counting of k-mers in dna sequences using a bloom filter. BMC bioinformatics **12**(1), 333 (2011)

[23] Roy, R.S., Bhattacharya, D., Schliep, A.: Turtle: Identifying frequent k-mers with cache-efficient algorithms. Bioinformatics **30**(14), 1950–1957 (2014)

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

[25] Mcvicar, N., Lin, C.-C., Hauck, S.: K-mer counting using bloom filters with an fpga-attached hmc. In: Field-Programmable Custom Computing Machines (FCCM), 2017 IEEE 25th Annual International Symposium On, pp. 203–210 (2017). IEEE

[26] Fan, L., Cao, P., Almeida, J., Broder, A.Z.: Summary cache: A scalable wide-area web cache sharing protocol. IEEE/ACM Trans. Netw. **8**(3), 281–293 (2000). https://doi.org/10.1109/90.851975

[27] Marinka Zitnik, S.M. Rok Sosič, Leskovec, J.: BioSNAP Datasets: Stanford Biomedical Network Dataset Collection. http://snap.stanford.edu/biodata (2018)

[28] Fan, B., Andersen, D.G., Kaminsky, M., Mitzenmacher, M.D.: Cuckoo filter: Practically better than bloom. In: Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies. CoNEXT '14, pp. 75–88. Association for Computing Machinery, New York, NY, USA (2014). https://doi.org/10.1145/2674005.2674994

[29] GitHub - efficient/cuckoofilter. https://github.com/efficient/cuckoofilter. (Accessed on 10/05/2021)

[30] GitHub - jvirkki/libbloom: A simple and small bloom filter implementation in plain C. https://github.com/jvirkki/libbloom. (Accessed on 21/05/2021)

[31] Graf, T.M., Lemire, D.: Xor filters: Faster and smaller than bloom and cuckoo filters. ACM J. Exp. Algorithmics **25** (2020). https://doi.org/10.1145/3376122

[32] GitHub - FastFilter/xor_singleheader: Header-only Xor Filter library. https://github.com/FastFilter/xor_singleheader. (Accessed on 02/05/2021)

[33] Nayak, S., Patgiri, R., Borah, A.: A survey on the roles of bloom filter in implementation of the named data networking. Computer Networks **196**, 108232 (2021). https://doi.org/10.1016/j.comnet.2021.108232

[34] Patgiri, R., Nayak, S., Borgohain, S.K.: Role of bloom filter in big data research: A survey. CoRR **abs/1903.06565** (2019)

[35] Nayak, S., Patgiri, R., Borah, A.: A survey on the roles of bloom filter in implementation of the named data networking. Computer Networks **196**, 108232 (2021). https://doi.org/10.1016/j.comnet.2021.108232

[36] Patgiri, R., Nayak, S., Muppalaneni, N.B.: Is bloom filter a bad choice for security and privacy? In: 2021 International Conference on Information Networking (ICOIN), pp. 648–653 (2021). https://doi.org/10.1109/ICOIN50884.2021.9333950

[37] Nayak, S., Patgiri, R.: A review on role of bloom filter on dna assembly. IEEE Access **7**, 66939–66954 (2019). https://doi.org/10.1109/ACCESS.2019.2910180

[38] Appleby, A.: Murmur Hashing. Retrieved from https://sites.google.com/site/murmurhash/ (2010)

[39] Patgiri, R., Nayak, S., Borgohain, S.K.: rdbf: A r-dimensional bloom filter for massive scale membership query. Journal of Network and Computer Applications **136**, 100–113 (2019). https://doi.org/10.1016/j.jnca.2019.03.004

[40] Bonomi, F., Mitzenmacher, M., Panigrah, R., Singh, S., Varghese, G.: Beyond bloom filters: from approximate membership checks to approximate state machines. ACM SIGCOMM Computer Communication Review **36**(4), 315–326 (2006). ACM

[41] Einziger, G., Friedman, R.: Tinyset—an access efficient self adjusting bloom filter construction. IEEE/ACM Transactions on Networking **25**(4), 2295–2307 (2017). https://doi.org/10.1109/TNET.2017.2685530

[42] Pagh, R., Rodler, F.F.: Cuckoo hashing. Journal of Algorithms **51**(2), 122–144 (2004)

[43] UCSC Genome Browser Downloads. http://hgdownload.cse.ucsc.edu/downloads.html. (Accessed on 15/03/2021)

[44] Goh, K.-I., Cusick, M.E., Valle, D., Childs, B., Vidal, M., Barabási, A.-L.: The human disease network. Proceedings of the National Academy of Sciences **104**(21), 8685–8690 (2007) https://arxiv.org/abs/https://www.pnas.org/content/104/21/8685.full.pdf. https://doi.org/10.1073/pnas.0701361104

[45] Gilbert-Diamond, D., Moore, J.H.: Analysis of gene-gene

interactions. Current protocols in human genetics **70**(1), 1–14 (2011)