cture(0,0)(-30,0)10 (-30,-5)(0,1)10 (-35,0)(1,0)30 (0,30)10 (-5,30)(1,0)10 (0,35)(0,-1)30 picturepicture(0,0)(30,0)10 (30,-5)(0,1)10 (35,0)(-1,0)30 (0,30)10

Published online DD Month 20YY *Nucleic Acids Research, 20YY, Vol. 00, No. 00* **1–21**
doi:10.1093/nar/gkn000

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

Sabuzima Nayak [1,*], **and Ripon Patgiri** [2 *]

[1]Corresponding Author and National Institute of Technology Silchar

Received Month DD, 20YY; Revised Month DD, 20YY; Accepted Month DD, 20YY

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 have proposed a novel Bloom Filter for biological networks, called BionetBF, to provide fast membership identification of the biological network edges or paired biological data. BionetBF is capable of executing millions of operations within a second on datasets having millions of paired biological data while occupying 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. BionetBF 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.

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, genetic regulatory networks, and genetic interaction networks. Protein-protein interaction networks present transient and stable interactions between the proteins where the interactions are undirected by nature (2). 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. 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 (3). 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 (4). Biological networks unfold prominent relationships, patterns, and properties of the whole system, which is difficult in a univariate analysis. Biological network is a great help in drug discovery. Typically, the commercial availability of a drug takes more than total 7 years including 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 interactions among the proteins result in cellular and molecular mechanisms that correspondingly determine an organism's health. Thus, the study on biological network of protein-protein interaction helps in determining the protein related to the diseases (5).

*To whom correspondence should be addressed: Sabuzima Nayak and Ripon Patgiri, National Institute of Technology Silchar, Cachar-788010, Assam, India, Email: sabuzima_rs@cse.nits.ac.in and ripon@cse.nits.ac.in

© 20YY The Author(s)
This is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (http://creativecommons.org/licenses/by-nc/2.0/uk/) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.

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 (6). 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 (7).

Analysis of large networks is predominantly computationally intractable (8). 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 (9). The biological network is dynamic in nature (10); 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, lots of information is missing. 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 them being noisy. Thus, it demands highly efficient data structures with low memory footprints.

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 (11).*

Bloom Filter (12) is an approximate set membership filtering data structures which is defined in Definition 1. It is extremely popular in diverse domains, particularly, Big Data, IoT, Cloud Computing, networking, security, database, Bioinformatics, 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 using a low memory footprint, but it has false positive probability issues. Nevertheless, the false positive probability can be mitigated by designing a new Bloom Filter.

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 network. BionetBF can perform insertion and query operations on millions of sequences within seconds. We have conducted extensive experiments on BionetBF using 12 generated datasets. It exhibits higher performance while maintaining a low false positive probability of 0.001. Moreover, the performance of BionetBF is measured using multiple parameters. BionetBF is compared with the other 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 is 6.6× larger memory size than BionetBF and Libbloom 4.7× larger memory size than BionetBF. The performance of BionetBF is measured using three biological network datasets, where BionetBF demonstrates higher efficiency than Libbloom and Cuckoo Filter.

PRELIMINARIES AND RELATED WORK

Bloom Filter

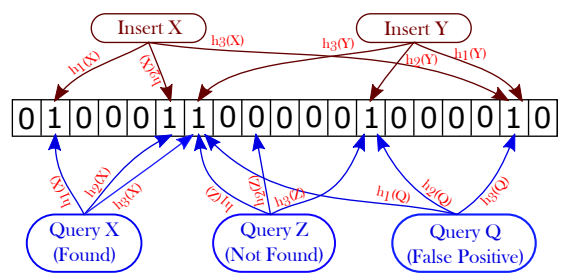


Figure 1. Architecture of Standard Bloom Filter using three hash functions.

Bloom Filter (12) is a bitmap array consisting of bits. Let $\mathbb{B}$ be the 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 (11). Bloom Filter calculates the optimal number of hash functions using Equation **1**.

$$k=\frac{m}{n}ln2 \quad \textbf{(1)}$$

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. $True$ 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 1 illustrates the bit array of a standard Bloom Filter. The Bloom Filter in the Figure 1 has $k=3$, i.e., $h_1()$, $h_2()$, and $h_3()$, and X, Y, Z and Q are three words. Initially the bit array of Bloom Filter is set to 0. The words X and Y are inserted. The input word X is hashed by three hash functions. The hash value gives the bit location as pointed in the figure. Three hash functions give three bit locations which are set to 1. Similar procedure is followed to insert word 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 word X gives True response whereas query of word Z gives False response. In case of query word Z, out of three bit locations, a bit location is 0; hence, Bloom Filter returns False.

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 (13) 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 Bloom Filter $\mathbb{B}$. Modulus operation gives three bit locations which are set to 1. Algorithm 2 query a word $Word$ into Bloom Filter $\mathbb{B}$. 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 Bloom Filter $\mathbb{B}$. 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.

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 1, a word Q is queried to the Bloom Filter $\mathbb{B}$, 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 both word X and Y as illustrated in Figure 1. 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 for both X and Y. Hence, after deletion of X, if Y is queried then $\mathbb{B}$ returns False. Even if the word Y is present in the Bloom Filter, it does not recognise the word. Therefore, the delete operations are restricted in standard Bloom Filter. Thus, Bloom Filter variants give more focus to reduce the false positive probability.

To measure false positive probability, the m is the number of bits in the bit array. The probability of a particular slot is not set to 1 by a specific hash function is $(1-\frac{1}{m})$. The probability of that particular slot is not set to 1 by k hash functions is given by Equation (**2**) (14).

$$(1-\frac{1}{m})^k \tag{2}$$

We know that

$$e^{-1}=\lim_{\infty}(1-\frac{1}{m})^m \tag{3}$$

Substituting Equation (**2**) by Equation (**3**), we get Equation (**4**).

$$(1-\frac{1}{m})^k=((1-\frac{1}{m})^m)^{k/m}\approx e^{-k/m} \tag{4}$$

If we insert n items into the bit array, then probability of that particular bit is still not set to 1 is given by Equation (**5**).

$$(1-\frac{1}{m})^{nk}\approx e^{-kn/m} \tag{5}$$

The probability of that particular bit is set to 1 is given by Equation (**6**).

$$(1-(1-\frac{1}{m})^{nk})\approx 1-e^{-kn/m} \tag{6}$$

The probability of all slots of μ bit array to be 1 is given in the Equation (**7**).

$$\varepsilon=(1-e^{-kn/m})^k \tag{7}$$

Equation (**7**) gives us the false positive probability. However, the value of k must be optimal for a certain number of inputs n; for instance, a large value of k increases the false positive probability, and also, a small value of k increases the false positive probability. Therefore, the value of k must be optimal to reduce the false positive probability.

$$k=\frac{m}{n}ln\ 2 \tag{8}$$

Equation (**8**) gives optimal false positive probability for given m bits array and n input items (15). Now, the value of k is replaced in Equation (**7**), we get Equation (**9**).

$$\varepsilon=\left(1-e^{-(\frac{m}{n}ln2)\frac{n}{m}}\right)^{\frac{m}{n}ln2} \tag{9}$$

Taking ln both side of Equation (**9**), we get Equation (**10**).

$$\begin{aligned} ln\ \varepsilon &= -\frac{m}{n}(ln\ 2)^2 \\ m &= -\frac{n\ ln\ \varepsilon}{(ln\ 2)^2} \end{aligned} \tag{10}$$

Equation (**10**) gives us the required memory size in bits for a given number input items and a desired false positive probability.

cture(0,0)(-30,0)10 (-30,-5)(0,1)10 (-35,0)(1,0)30 (0,30)10 (-5,30)(1,0)10 (0,35)(0,-1)30 picturepicture(0,0)(30,0)10 (30,-5)(0,1)10 (35,0)(-1,0)30 (0,30)10

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

Algorithm 2 Query a word $Word$ in Bloom Filter using three hash functions.

```
procedure LOOKUP(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 using three 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
```

Cuckoo Filter

Cuckoo Filter (16) 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 (17), 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 storage. 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 its two locations; if it is present in any one location, then the Cuckoo Filter returns true; otherwise false.

MATERIALS AND METHODS

BionetBF is a novel Bloom Filter for membership identification of paired biological data. BionetBF is the modified 2DBF (18). 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.

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. 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 (13). 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

cture(0,0)(-30,0)10 (-30,-5)(0,1)10 (-35,0)(1,0)30 (0,30)10 (-5,30)(1,0)10 (0,35)(0,-1)30 picturepicture(0,0)(30,0)10 (30,-5)(0,1)10 (35,0)(-1,0)30 (0,30)10

Algorithm 4 Insertion of a paired biological data $(ID, Kmer)$ into BionetBF. The $\ll$ is left shift operation.

procedure IBIONETBF($\mathbb{B}_{x,y}$, ID, $Kmer$) ▷ x and y are dimensions of BionetBF
  $\mathcal{C}_{IDKmer} = \text{STRCAT}(ID, Kmer)$ ▷ Concatenating ID and Kmer
  for $k \leftarrow 1$ to 2 **do**
    if $k == 1$ **then**
      $Seed \leftarrow Seed_1$
    else
      $Seed \leftarrow Seed_2$
    end if
    $h_{IDKmer} = \mathcal{H}(\mathcal{C}_{IDKmer}, Seed)$ ▷ $\mathcal{H}()$ is murmur hash function
    $i = h_{IDKmer} \% \, x$
    $j = h_{IDKmer} \% \, y$
    $pos = h_{IDKmer} \% \, bits$
    $d = \mathbb{B}_{i,j}$
    $p = 1 \ll pos$
    $\mathbb{B}_{i,j} = d \; OR \; p$
  end for
end procedure

value using the dimensions of the BionetBF. The modular operations gives 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 5 demonstrates the query operation on BionetBF. The steps till obtaining a location in BionetBF is the same for both insertion and query operation. 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.

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

To illustrate the above definitions, let $\mathcal{I} = \{(1, ATGA), (2, CGAG), (3, AGAG), (4, CAGC), (5, CATT), \ldots, (10, TTAG)\}$
Then, $\mathcal{S} = \{(1, ATGA), (2, CGAG), (3, AGAG), (4, CAGC), (5, CATT), \ldots, (10, TTAG)\}$

Algorithm 5 Query a paired biological data $(ID, Kmer)$ into BionetBF. The $\gg$ is right shift operation.

procedure QBIONETBF($\mathbb{B}_{x,y}$, ID, $Kmer$)
  $\mathcal{C}_{IDKmer} = \text{STRCAT}(ID, Kmer)$ ▷ Concatenating ID and Kmer
  for $k \leftarrow 1$ to 2 **do**
    if $k == 1$ **then**
      $Seed \leftarrow Seed_1$
    else
      $Seed \leftarrow Seed_2$
    end if
    $h_{IDKmer} = \mathcal{H}(\mathcal{C}_{IDKmer}, Seed)$ ▷ $\mathcal{H}()$ is murmur hash function
    $i = h_{IDKmer} \% \, x$
    $j = h_{IDKmer} \% \, y$
    $pos = h_{IDKmer} \% \, bits$
    $d = \mathbb{B}_{i,j}$
    $p = 1 \gg pos$
    $r = d \oplus p$
    $t = r \; AND \; p$
    if $(t = 0) \; AND \; (d \neq 0))$ **then**
      $flag \leftarrow 1$
    else
      $flag \leftarrow 0$
    end if
  end for
  if $flag == 1$ **then**
    return true
  else
    return false
  end if
end procedure

$\mathcal{D} = \{(11, TACT), (12, GCTC), (13, TCTC), (14, GTCG), (15, GTAA), \ldots, (20, AATC)\}$ 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.

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 (19)). 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.

cture(0,0)(-30,0)10 (-30,-5)(0,1)10 (-35,0)(1,0)30 (0,30)10 (-5,30)(1,0)10 (0,35)(0,-1)30 picturepicture(0,0)(30,0)10 (30,-5)(0,1)10 (35,0)(-1,0)30 (0,30)10

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

Biological Network Dataset: For the experiments, three biological network datasets are considered: Chemical-Gene, Gene-Disease, and Gene-Gene (Downloaded from (20)). 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 (21). In the Gene-Gene dataset, both the biological data is gene ID. The interaction between gene and gene is also called Epistasis (22).

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 2. File size of biological network Dataset.

Performance Parameters

Let $\mathcal{FPP}$, $\mathcal{FNP}$, $\mathcal{T}$, $\mathcal{N}$, $\mathcal{B}$, and n be the false positive probability, false negative probability, time taken (in seconds), number of operations (in million), Bloom Filter's size (in bits), and number of input sequences, respectively. Equation (**11**), (**12**), (**13**), (**14**) and (**15**) 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.

$$Accuracy = 1 - (\mathcal{FPP} + \mathcal{FNP}) \quad \textbf{(11)}$$

$$Million\ Operations\ per\ Second\ (MOPS) = \frac{\mathcal{N}}{\mathcal{T}} \quad \textbf{(12)}$$

$$Second\ per\ Operations\ (SPO) = \frac{\mathcal{T}}{\mathcal{N}} \quad \textbf{(13)}$$

$$Megabytes\ Per\ Second\ (MBPS) = \frac{\mathcal{S}}{\mathcal{T}} \quad \textbf{(14)}$$

$$Bits\ per\ Sequences\ (BPS) = \frac{\mathcal{B}}{n} \quad \textbf{(15)}$$

RESULTS

We have conducted extensive series of tests to validate the performance of BionetBF using diverse datasets as given in Table 1 and 2. The experimental analysis shows the reduction of the memory footprint of BionetBF to construct a data structure for biological network data. Moreover, we prove two major concerning points regarding biological network through conducting rigorous experiments using BionetBF: 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.

cture(0,0)(-30,0)10 (-30,-5)(0,1)10 (-35,0)(1,0)30 (0,30)10 (-5,30)(1,0)10 (0,35)(0,-1)30 picturepicture(0,0)(30,0)10 (30,-5)(0,1)10 (35,0)(-1,0)30 (0,30)10

Selecting k for BionetBF

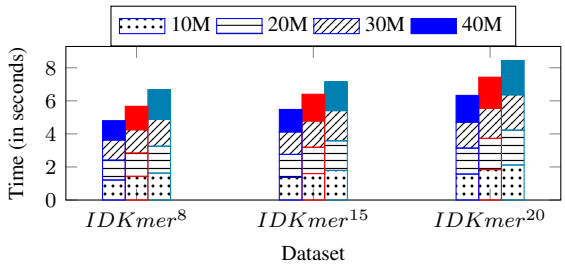


Figure 2. Insertion Time (BionetBF $k=1$ vs 2 vs 3): Comparison of insertion time among BionetBF having $k=1$, 2 and 3. 10M: 10 million, 20M: 20 million, 30M: million, 40M: 40 million number of IDKmers inserted. Lower value is better.

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 based on 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 $BionetBF_{k=3}$.

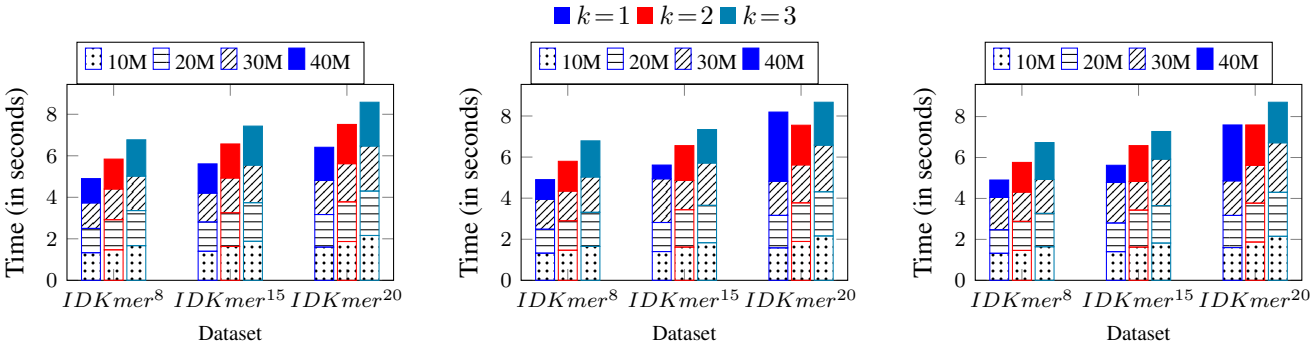
Figure 2 and Figure 3 illustrate the time taken by $BionetBF_{k=1}$, $BionetBF_{k=2}$ and $BionetBF_{k=3}$ for insertion and query operation respectively. In both figures, each bar is divided into four segments; (consider from bottom) the 1st, 2nd, 3rd, and 4th segment represents the 10 million, 20 million, 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 secs on average and from 2 to 3 the insertion time increases by 0.57 secs on average. Figure 3a, Figure 3b, and Figure 3c exhibit the query time for Same Set, Mixed Set and Disjoint Set, respectively. In Figure 3a, i.e., 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 $IDKmer^{8}_{10}$. Moreover, 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 secs on average and from 2 to 3 the query time increases by 0.59 secs on average. In Figure 3b, i.e., 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 $IDKmer^{8}_{10}$. Furthermore, 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 secs on average and from 2 to 3 the query time increases by 0.6 secs on average. In Figure 3c, i.e., 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 $IDKmer^{8}_{10}$. In addition, the maximum time taken by $BionetBF_{k=1}$, $BionetBF_{k=2}$ and $BionetBF_{k=3}$ is 7.6sec, 7.6sec, and 8.7sec, respectively for $IDKmer^{20}_{40}$. Increase of the k value from 1 to 2, the query time for Disjoint Set increases by 0.43 secs on average and from 2 to 3 the query time increases by 0.61 secs on average.

Figure 4 evince the comparison of $BionetBF_{k=1}$, $BionetBF_{k=2}$ and $BionetBF_{k=3}$ based on FPP. Figure 4a and Figure 4b 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 4a), $BionetBF_{k=1}$ 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 4b), $BionetBF_{k=1}$ gives more than 0.001 FPP in $IDKmer^{20}$, $BionetBF_{k=2}$ gives 0.0017 FPP in $IDKmer^{20}_{40}$, and $BionetBF_{k=3}$ has zero FPP in all 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.

The accuracy in Same Set is $1-(FNP+FPP)$ where FNP is false negative probability. BionetBF manifests 100% accuracy in all IDKmer dataset in Same Set because the BionetBF has zero false negative. In case of Mixed Set and Disjoint Set as delineate in Figure 5a and Figure 5b, respectively, $BionetBF_{k=1}$, $BionetBF_{k=2}$ and $BionetBF_{k=3}$ exhibited 100% accuracy for $IDKmer^8$ and $IDKmer^{15}$. With $IDKmer_{40}$ dataset, $BionetBF_{k=1}$ demonstrates the lowest accuracy; however, it is 99% in the Mixed Set as shown in Figure 5a and 98% in the Disjoint Set as shown in Figure 5b.

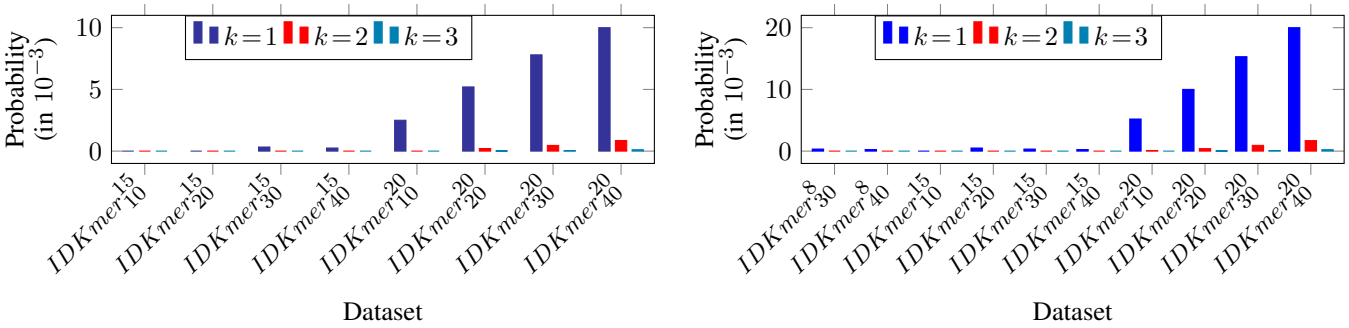
Figure 6 highlights comparison among $BionetBF_{k=1}$, $BionetBF_{k=2}$ and $BionetBF_{k=3}$ based on the performance of insertion operation. Figure 6a demonstrates the comparison based on insertion MOPS. The highest SPO of $BionetBF_{k=1}$, $BionetBF_{k=2}$ and $BionetBF_{k=3}$ is 8.31 ($IDKmer^{8}_{40}$), 7.06 ($IDKmer^{8}_{30}$), and 6.16 ($IDKmer^{8}_{30}$), respectively, whereas the lowest MOPS of 6.31 ($IDKmer^{20}_{40}$), 5.36 ($IDKmer^{20}_{20}$), and 4.72 ($IDKmer^{20}_{30}$), respectively. Figure 6b shows the comparison of performance based on insertion SPO. The highest SPO of $BionetBF_{k=1}$, $BionetBF_{k=2}$ 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 6c 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

cture(0,0)(-30,0)10 (-30,-5)(0,1)10 (-35,0)(1,0)30 (0,30)10 (-5,30)(1,0)10 (0,35)(0,-1)30 picturepicture(0,0)(30,0)10 (30,-5)(0,1)10 (35,0)(-1,0)30 (0,30)10



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

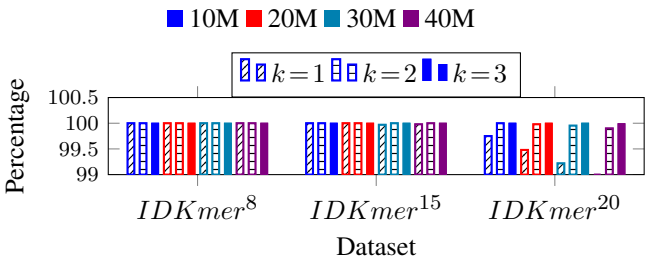
Figure 3. Query Time (BionetBF $k=1$ *vs* 2 *vs* 3): Comparison of time taken by BionetBF having $k=1,2$ *and* 3 for the Same Set, Mixed Set and Disjoint Set query operation. 10M: 10 million, 20M: 20 million, 30M: million, and 40M: 40 million number of IDKmer queries. Lower value is better.



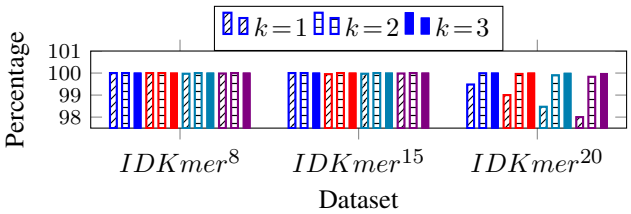
(a) Mixed Set (b) Disjoint Set

Figure 4. 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.

$BionetBF_{k=3}$ 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 7 delineate the comparison among $BionetBF_{k=1}$, $BionetBF_{k=2}$ and $BionetBF_{k=3}$ based on the performance of query operation using MOPS. Figure 7a, Figure 7b and Figure 7c demonstrate on the query MOPS for Same Set, Mixed Set and Disjoint Set, respectively. In Same Set (Figure 7a), the highest MOPS of $BionetBF_{k=1}$, $BionetBF_{k=2}$ 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 7b), $BionetBF_{k=1}$, $BionetBF_{k=2}$ 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 7c), $BionetBF_{k=1}$, $BionetBF_{k=2}$ and $BionetBF_{k=3}$ has the highest MOPS is 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 8 shows the comparison of performance based on SPO for query operation. Figure 8a, Figure 8b and Figure 8c illustrate the query SPO for Same Set, Mixed Set and Disjoint Set, respectively. In Same Set (Figure 8a), 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 8b), 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 8c), 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 SPO is $1.23E-07$ ($IDKmer^{8}_{30}$), $1.44E-07$ ($IDKmer^{8}_{30}$), and $1.654E-07$ ($IDKmer^{8}_{20}$), respectively. Figure 9 demonstrates the differentiation among $BionetBF_{k=1}$, $BionetBF_{k=2}$ and $BionetBF_{k=3}$ using performance based on MBPS for query operation. Figure 9a, Figure 9b and Figure 9c demonstrate

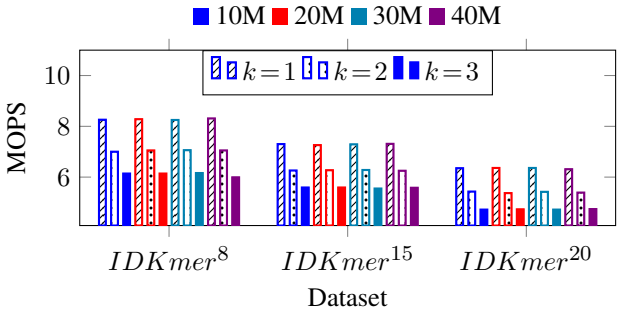


(a) Mixed Set

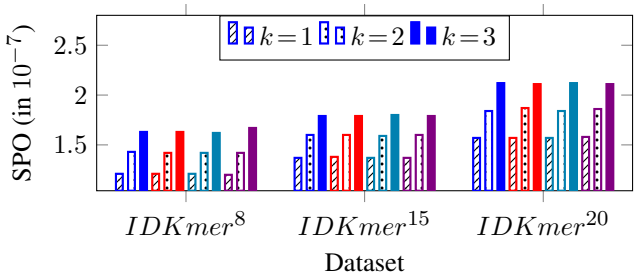


(b) Disjoint Set

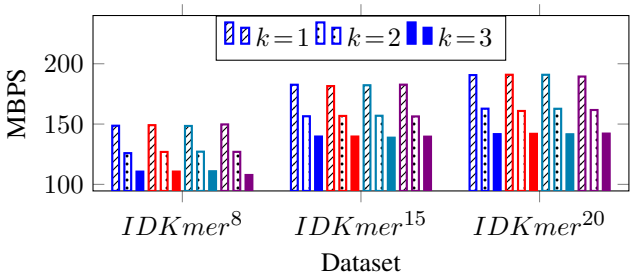
Figure 5. Accuracy (BionetBF having $k=1,2$ *and* 3): Comparison of accuracy of querying Mixed Set and Disjoint Set to BionetBF having $k=1,2$ *and* 3. Accuracy of Same Set is 100%. Higher value is better. 10M: 10 million, 20M: 20 million, 30M: million, and 40M: 40 million number of IDKmer queries.



(a) Million operation per second (MOPS)



(b) Seconds per operation (SPO)



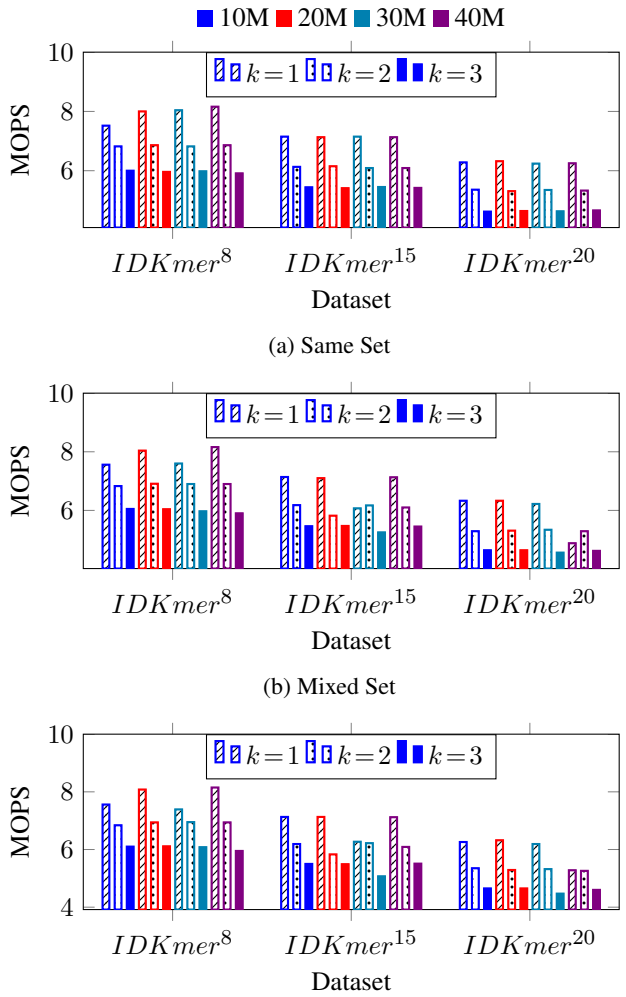
(c) Megabytes per second (MBPS)

Figure 6. 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.

the query in MBPS for Same Set, Mixed Set and Disjoint Set, respectively. Figure 9a 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 ($IDKmer_{10}^{20}$), 160.86 ($IDKmer_{10}^{20}$), and 139.66 ($IDKmer_{40}^{20}$), whereas, the lowest MBPS is 135.34 ($IDKmer_{20}^{8}$), 122.7 ($IDKmer_{10}^{8}$), and 106.46 ($IDKmer_{40}^{8}$), respectively. Figure 9b 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 ($IDKmer_{10}^{20}$), 160.06 ($IDKmer_{30}^{20}$), and 139.34 ($IDKmer_{20}^{20}$), while, the lowest MBPS is 136.05 ($IDKmer_{30}^{8}$), 122.95 ($IDKmer_{10}^{8}$), and 106.16 ($IDKmer_{40}^{8}$), respectively. Figure 9c 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 ($IDKmer_{10}^{20}$), 160.6 $IDKmer_{10}^{20}$), and 139.5 ($IDKmer_{20}^{20}$), and the lowest MBPS is 133 ($IDKmer_{30}^{8}$), 123.2 ($IDKmer_{10}^{8}$), and 107.02($IDKmer_{40}^{8}$), respectively.

We observe from Figure 5, Figure 7, Figure 8, and Figure 9 that the performance of $BionetBF_{k=3}$ is better than $BionetBF_{k=2}$ and $BionetBF_{k=2}$ 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.

Experimentation with higher k values: 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 $IDKmer_{40}^{20}$. Figure 10 presents the insertion time of BionetBF having $4 \leq k \leq 10$. The $BionetBF_{k=4}$ takes 9.57 sec whereas $BionetBF_{k=10}$ takes 17.21 sec. 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 11 depicts the query time of BionetBF having $4 \leq k \leq 10$. 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. 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

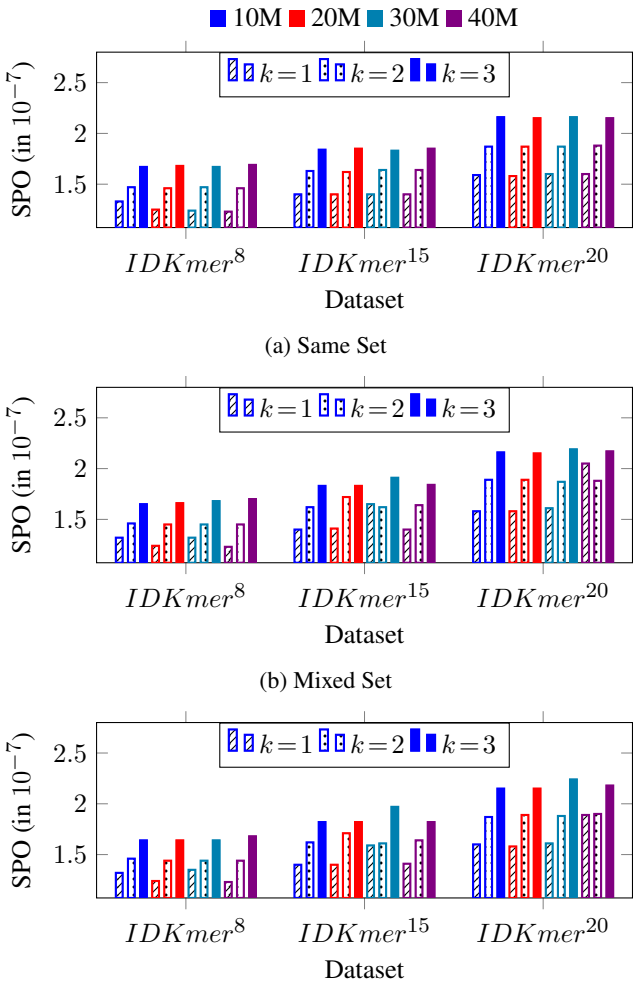


(a) Same Set

(b) Mixed Set

(c) Disjoint Set

Figure 7. 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.



(a) Same Set

(b) Mixed Set

(c) Disjoint Set

Figure 8. 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.

other words, it takes less than a minute. Figure 12 exposes the FPP of the BionetBF having $4 \leq k \leq 10$. BionetBF with $k=6$ and higher, the FPP becomes zero. 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 using less time duration for operation.

BionetBF

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

Table 3. 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 13 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 14 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 the case of the 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 $IDKmer^{8}$, $IDKmer^{15}$,

cture(0,0)(-30,0)10 (-30,-5)(0,1)10 (-35,0)(1,0)30 (0,30)10 (-5,30)(1,0)10 (0,35)(0,-1)30 picturepicture(0,0)(30,0)10 (30,-5)(0,1)10 (35,0)(-1,0)30 (0,30)10

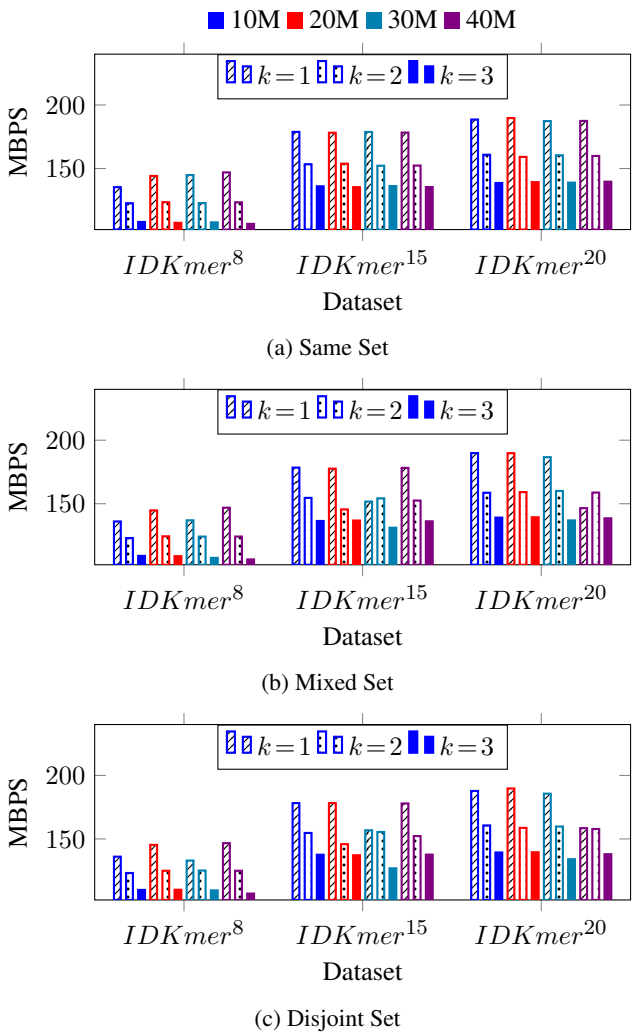


Figure 9. Query MBPS (BionetBF having $k=1,2$ *and* 3): Comparison of performance among BionetBF having $k=1,2$ *and* 3 based on Megabyte per second (MBPS) for query operations in case of (a) Same Set, (b) Mixed Set, and (c) Disjoint Set. Higher is better. 10M: 10 million, 20M: 20 million, 30M: million, and 40M: 40 million number of IDKmer queries.

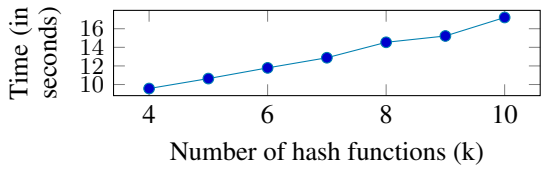


Figure 10. Insertion Time (BionetBF having $4 \leq k \leq 10$): Comparison of insertion time among BionetBF having $4 \leq k \leq 10$. The dataset is IDKmer with sequence length 20 with 40 million IDKmers. Lower value is better.

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

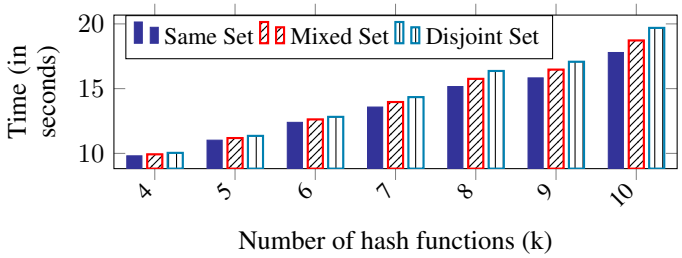


Figure 11. Query Time (BionetBF having $4 \leq k \leq 10$): Comparison of query time among BionetBF having $4 \leq k \leq 10$. The dataset is IDKmer with sequence length 20 with 40 million queries. Lower value is better.

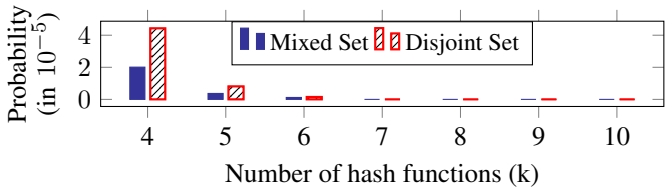


Figure 12. False Positive Probability (BionetBF having $4 \leq k \leq 10$): Comparison of false positive probability of querying Mixed Set and Disjoint Set to BionetBF having $4 \leq k \leq 10$. The dataset is IDKmer with sequence length 20 with 40 million queries. Lower value is better.

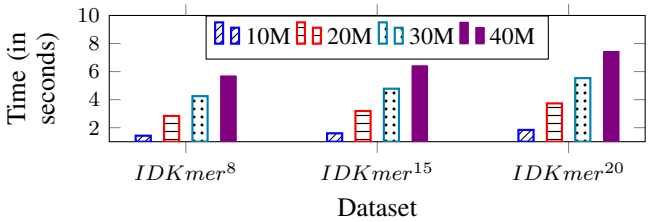


Figure 13. Insertion Time (BionetBF): Comparison of time taken by insertion operation by BionetBF in case of IDKmer having sequence length 8, 15 and 20. Lower value is better. 10M: 10 million, 20M: 20 million, 30M: million, and 40M: 40 million number of IDKmers inserted.

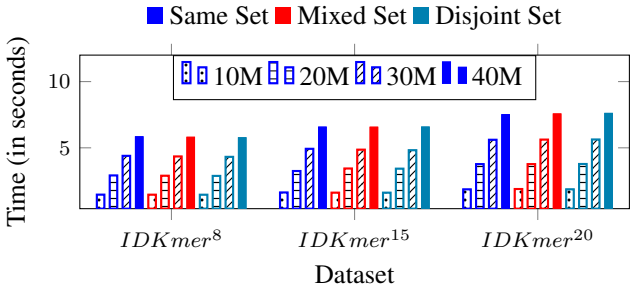


Figure 14. Query Time (BionetBF): Comparison of time taken by BionetBF; querying Same Set, Mixed Set and Disjoint Set. Lower is better. 10M: 10 million, 20M: 20 million, 30M: million, and 40M: 40 million number of IDKmer queries.

querying Disjoint Set takes 1.46 sec for $IDKmer^{8}_{10}$ and 7.6 sec for $IDKmer^{20}_{40}$. 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 $IDKmer^{8}$,

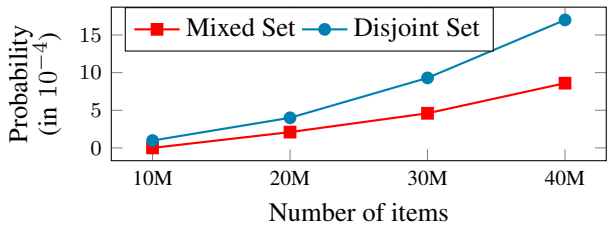
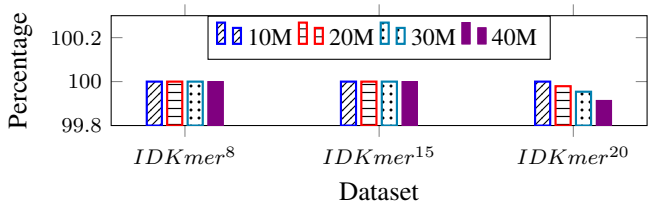
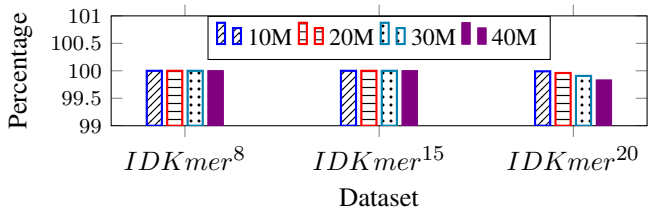


Figure 15. False Positive Probability (BionetBF): Comparison of false positive probability of BionetBF; querying Mixed Set and Disjoint Set. The dataset is IDKmer having sequence length 20. The false positive probability of Mixed Set and Disjoint Set of IDKmer having sequence length 8 and 15 is zero. 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

Figure 16. 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.

$IDKmer^{15}$, 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 15 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 false negative probability 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 16 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 16a and Figure 16b 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% respectively. Thus, for a large data file, BionetBF has above 99% of accuracy. Figure 17 represents the performance of the insertion operation.

Figure 17a 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 17b evince SPO in which the highest SPO is 1.87 ($IDKmer^{20}_{20}$) and the lowest is 1.42 ($IDKmer^8_{30}$). Figure 17c depicts MBPS where the highest MBPS is 162.78 ($IDKmer^{20}_{10}$) and the lowest is 125.87 ($IDKmer^8_{10}$). Similarly, Figure 18 delineates the performance of the query operation. Figure 18a, Figure 18b, and Figure 18c 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 18d, Figure 18e, and Figure 18f 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 18g, Figure 18h, and Figure 18i 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 (**15**)) illustrated in Table 3. 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.

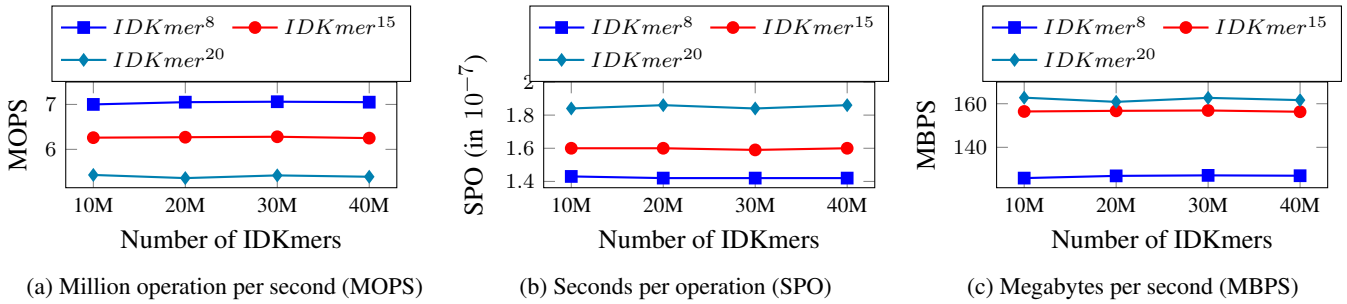
Comparison with Other Bloom Filters

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

Table 4. Details of Cuckoo Filter, Libbloom, and BionetBF. #Hash functions: Number of hash functions executed per operation.

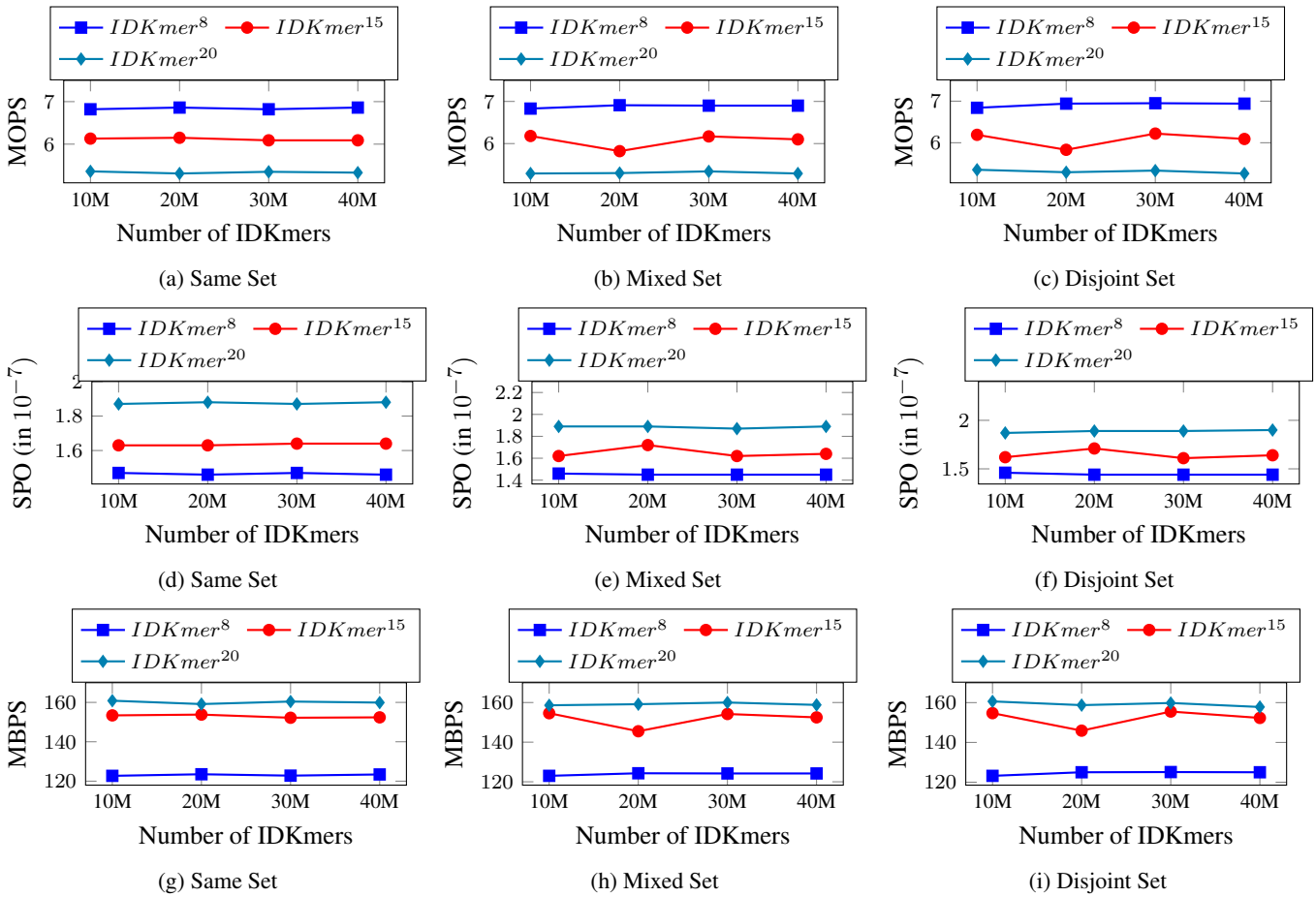
Number of IDKmers	Cuckoo Filter	Libbloom	BionetBF
10 million	80.53	57.51	10.85
20 million	40.27	28.76	5.43
30 million	26.84	19.17	3.61
40 million	20.13	14.38	2.71

Table 5. Comparison of memory usage among Cuckoo Filter, Libbloom, and BionetBF based on Bits Per Sequences (BPS).



(a) Million operation per second (MOPS) (b) Seconds per operation (SPO) (c) Megabytes per second (MBPS)

Figure 17. 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

Figure 18. 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.

In this section, BionetBF is compared with two other Bloom Filters: Cuckoo Filter (16) (Code available at (23)) and Libbloom (Code available at (24)). 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. Importantly, if Cuckoo Filter is executed with the same dataset multiple times, it gives different false positives. This experiment considers the least number of false positives after executing the Cuckoo Filter multiple times for a single dataset. The Libbloom is the

cture(0,0)(-30,0)10 (-30,-5)(0,1)10 (-35,0)(1,0)30 (0,30)10 (-5,30)(1,0)10 (0,35)(0,-1)30 picturepicture(0,0)(30,0)10 (30,-5)(0,1)10 (35,0)(-1,0)30 (0,30)10

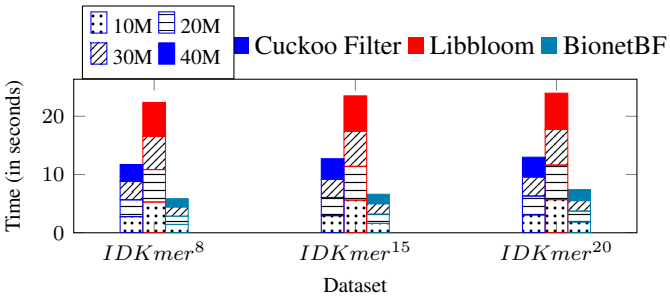


Figure 19. Insertion Time (Cuckoo Filter vs Libbloom vs BionetBF): Comparison of insertion time among Cuckoo Filter, Libbloom, and BionetBF. 10M: 10 million, 20M: 20 million, 30M: 30 million, 40M: 40 million number of IDKmers inserted.

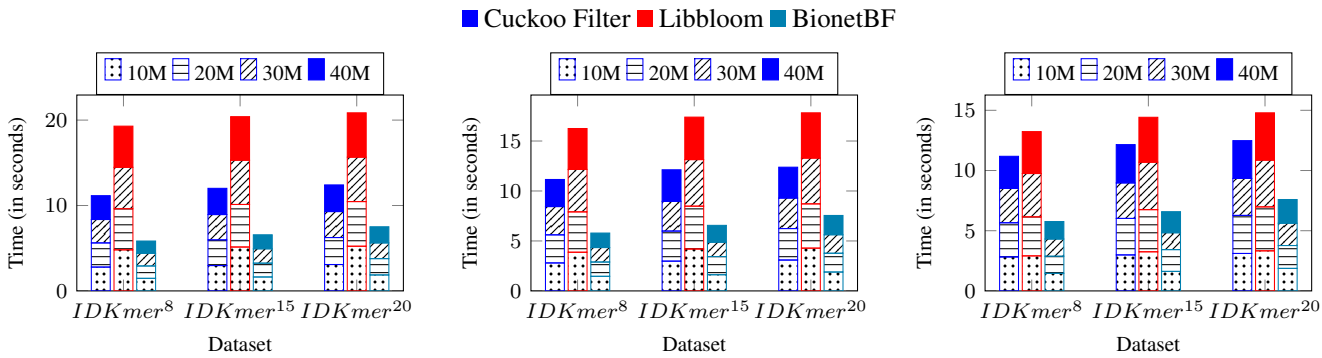
standard Bloom Filter (12). Similar to Cuckoo Filter, in the case of Libbloom, the total number of sequences considered is 40 million. Hence, the memory size is 71 MB and to achieve an FPP of 0.001; the Libbloom takes 10 hash functions. In the experiment, the filter size of BionetBF is 15MB. Similar 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 4 highlights the filter size and number of hash functions of Cuckoo Filter, Libbloom, and BionetBF, which are used in the experiment.

Figure 19 and Figure 20 show the comparison among Cuckoo Filter, Libbloom and BionetBF based on the time taken for insertion and query operation, respectively. Similar to Figure 2 and Figure 3, (considering from bottom) the 1st, 2nd, 3rd, and 4th segment of Figure 19 and Figure 20 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 19 and Figure 20. 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 secs 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 secs for $IDKmer^8$, $IDKmer^{15}$, and $IDKmer^{20}$, respectively. BionetBF takes 1.41, 1.6 and 1.86 secs more for $IDKmer^8$, $IDKmer^{15}$, and $IDKmer^{20}$, respectively with every increase in 10 million IDKmers. Figure 20a, Figure 20b, and Figure 20c 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 20a), the query time taken for $IDKmer^{20}_{40}$ by Cuckoo Filter, Libbloom and BionetBF is 12.4 sec, 20.84 sec, 7.5 sec, respectively, whereas for $IDKmer^8_{10}$ is 2.79 sec, 4.81 sec, and 1.47 sec. Likewise, in Mixed Set (Figure 20b), the query time taken for $IDKmer^{20}_{40}$ by Cuckoo Filter, Libbloom and BionetBF is 12.38 sec, 17.82 sec, 7.56 sec, respectively; while for $IDKmer^8_{10}$ is 2.79 sec, 3.87 sec, and 1.46 sec, respectively. In Disjoint Set (Figure 20c), the query time taken for $IDKmer^{20}_{40}$ by Cuckoo Filter, Libbloom and BionetBF is 12.47 sec, 14.78 sec, 7.6 sec, respectively, while for $IDKmer^8_{10}$ is 2.82 sec, 2.91 sec, and 7.6 sec, respectively. In case of query time, Libbloom takes the longest time, and BionetBF takes the least time. In the same way, with every increment of 10 million IDKmer queries, the query time Cuckoo Filter increases by on average 2.79, 3.04, and 3.12 secs 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 secs for the Disjoint Set of $IDKmer^8$, $IDKmer^{15}$, and $IDKmer^{20}$, respectively. BionetBF takes 1.43, 1.65 and 1.91 secs more for the Disjoint Set of $IDKmer^8$, $IDKmer^{15}$, and $IDKmer^{20}$, respectively with every increment of 10 million IDKmer queries.

Figure 21 and Figure 22 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 and 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 21), 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, then 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 (Figure22), 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. Similarly, BionetBF has zero FPP in the majority of Disjoint IDKmer datasets, whereas Libbloom has some FPP in the 10 million IDKmer datasets. Therefore, BionetBF exhibits superior performance with regard to insertion time, query time and FPP with a small filter size for high input IDKmer.

Figure 23 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 Cuckoo Filter and Libbloom are constructed with different architecture to achieve the desired FPP (i.e., 0.001).

Figure 24 delineate comparison among Cuckoo Filter, Libbloom and BionetBF based on the performance of insertion operation. Figure 24a 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 24b 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 $1.87E-07$



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

Figure 20. 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.

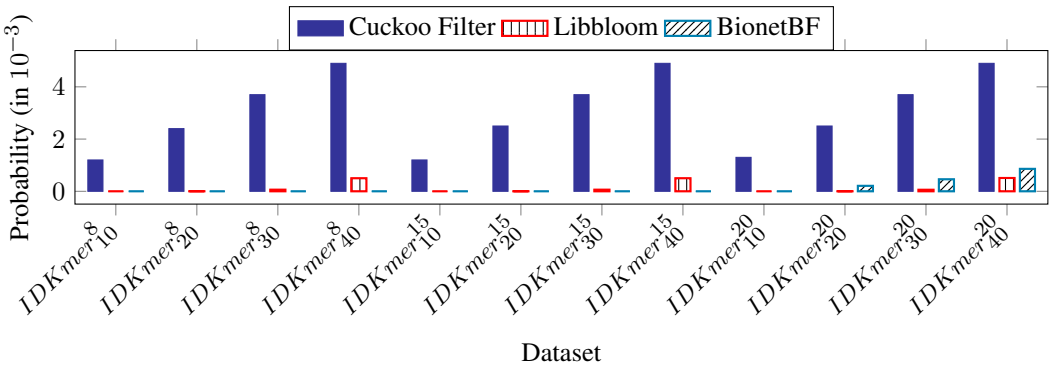


Figure 21. False Positive Probability (Libbloom vs Cuckoo Filter vs BionetBF): Comparison of false positive probability of querying Mixed Set to Libbloom, Cuckoo Filter and BionetBF.

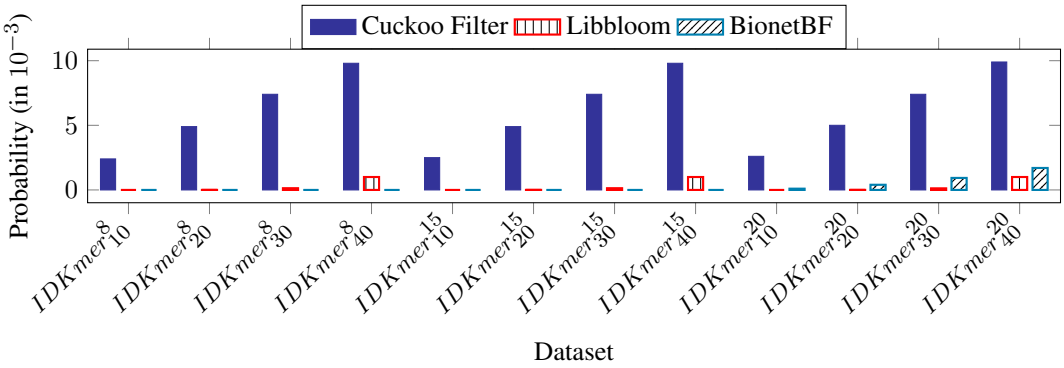
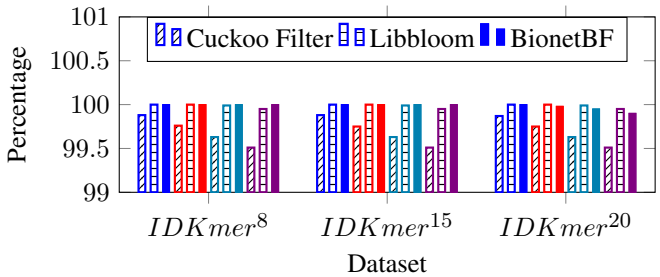
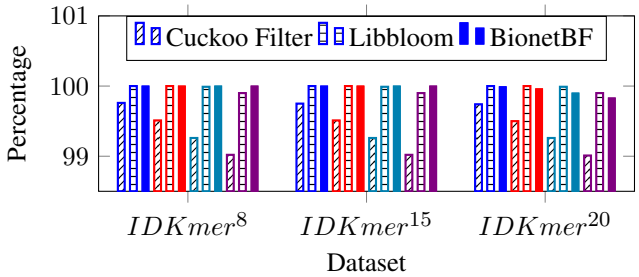


Figure 22. False Positive Probability (Libbloom vs Cuckoo Filter vs BionetBF): Comparison of false positive probability of querying Disjoint Set to Libbloom, Cuckoo Filter and BionetBF.

($IDKmer^{20}_{20}$) 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 24c 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 25, Figure 26, and Figure 27 highlight the analogy among Cuckoo Filter, Libbloom and BionetBF based on the performance of query operation. Figure 25 represent the query MOPS of

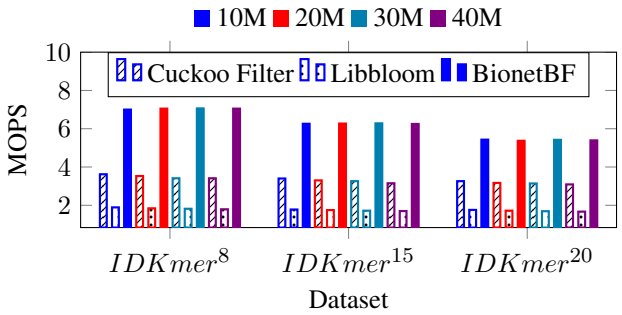


(a) Mixed Set

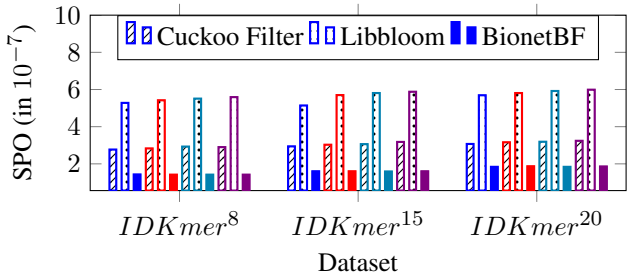


(b) Disjoint Set

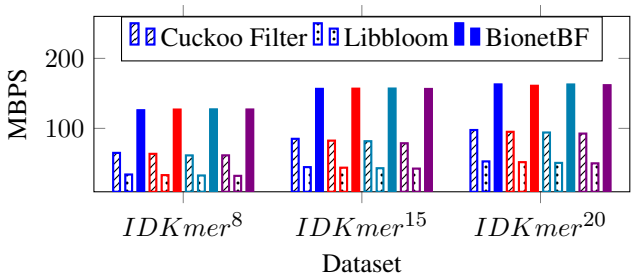
Figure 23. Accuracy (Cuckoo Filter vs Libbloom vs BionetBF): Comparison of accuracy of querying Same Set, Mixed Set and Disjoint Set to Libbloom, Cuckoo Filter, and BionetBF. Accuracy of Same Set is 100%. Higher value is better. 10M: 10 million, 20M: 20 million, 30M: million, and 40M: 40 million number of IDKmers queried.



(a) Million operation per second (MOPS)



(b) Seconds per operation (SPO)

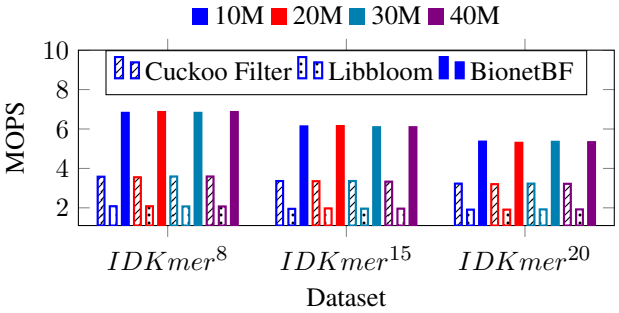


(c) Megabytes per second (MBPS)

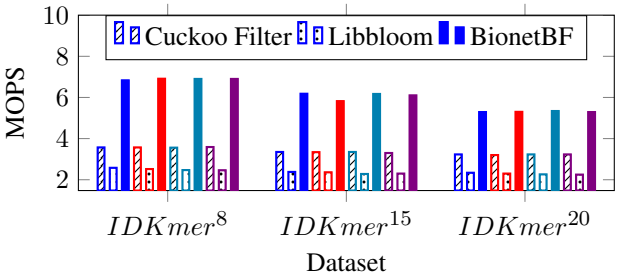
Figure 24. Performance of Insertion Operation (Cuckoo Filter vs Libbloom vs BionetBF): Comparison of performance among Cuckoo Filter, Libbloom, and BionetBF based on the (a) number of million insertion operations executed per second (MOPS), (b) time taken for execution of one insertion operation (SPO), 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.

Cuckoo Filter, Libbloom and BionetBF where Figure 25a, Figure 25b, and Figure 25c highlights the query MOPS for Same Set, Mixed Set and Disjoint Set respectively. In Same Set (Figure 25a), the highest MOPS shown by Cuckoo Filter, Libbloom and BionetBF is 359 ($IDKmer^8_{30}$), 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 25b), the highest MOPS of 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 of 3.2 ($IDKmer^{20}_{20}$), 2.25 ($IDKmer^{20}_{40}$), and 5.29 ($IDKmer^{20}_{10}$), respectively. In Disjoint Set (Figure 25c), the highest MOPS of 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 of 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 26 shows the comparison of performance based on SPO for query operation. Figure 26a, Figure 26b and Figure 26c illustrate the query SPO for Same Set, Mixed Set and Disjoint Set, respectively. In Same Set (Figure 26a), 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 26b), 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 26c), 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

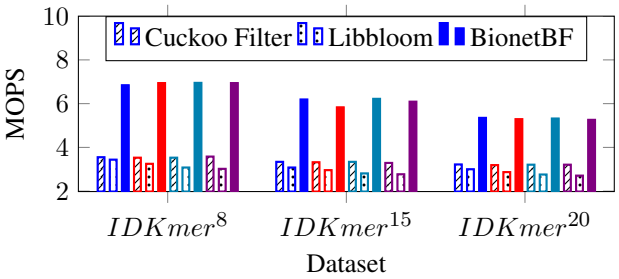
cture(0,0)(-30,0)10 (-30,-5)(0,1)10 (-35,0)(1,0)30 (0,30)10 (-5,30)(1,0)10 (0,35)(0,-1)30 picturepicture(0,0)(30,0)10 (30,-5)(0,1)10 (35,0)(-1,0)30 (0,30)10



(a) Same Set

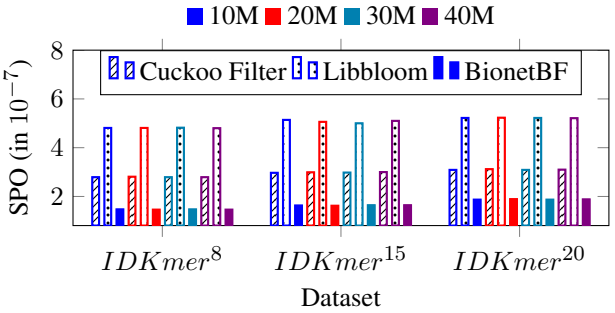


(b) Mixed Set

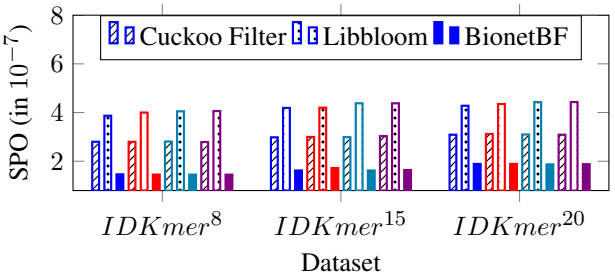


(c) Disjoint Set

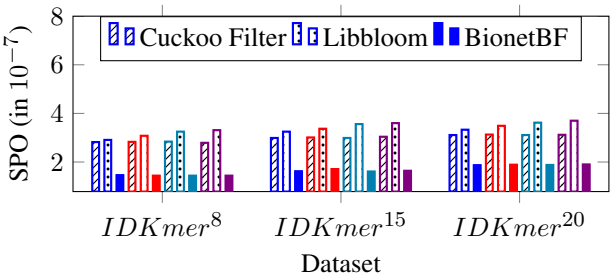
Figure 25. Query MOPS (Cuckoo Filter vs Libbloom vs BionetBF): Comparison of performance among Cuckoo Filter, Libbloom, and BionetBF based on the million query operations executed per second (MOPS) for (a) Same Set, (b) Mixed Set, and (c) Disjoint Set. Higher is better. 10M: 10 million, 20M: 20 million, 30M: million, and 40M: 40 million number of IDKmers queried.



(a) Same Set



(b) Mixed Set



(c) Disjoint Set

Figure 26. Query SPO (Cuckoo Filter vs Libbloom vs BionetBF): Comparison of performance among Cuckoo Filter, 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.

$1.44E-07$ ($IDKmer^8_{30}$), 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 27 demonstrates the differentiation among Cuckoo Filter, Libbloom, and BionetBF using performance based on query MBPS. Figure 27a, Figure 27b and Figure 27c illuminate the query MBPS for Same Set, Mixed Set and Disjoint Set, respectively. Figure 27a 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 27b 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 27c 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

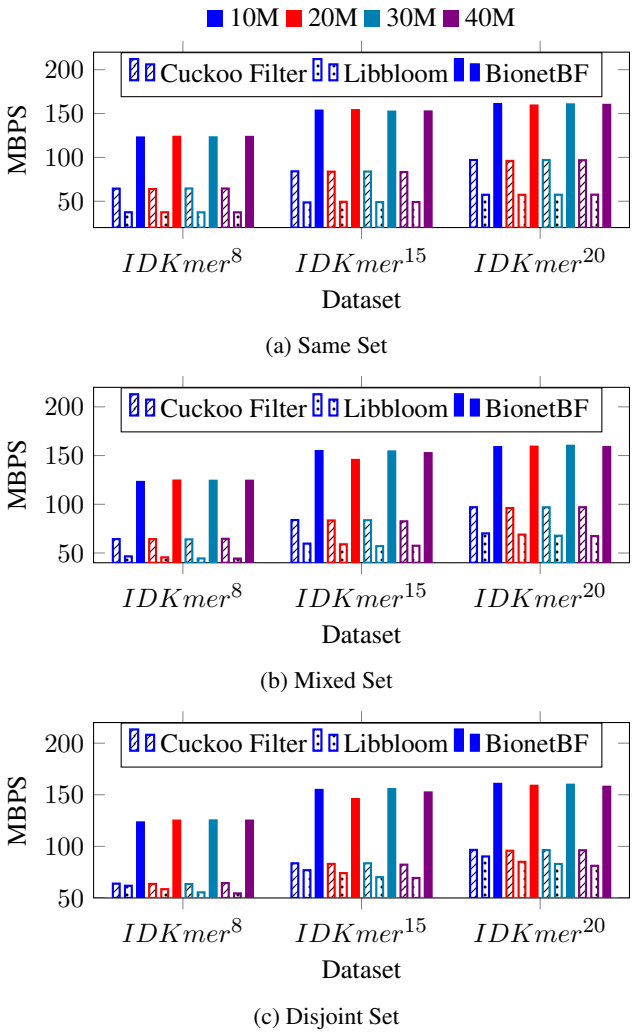


Figure 27. Query MBPS (Cuckoo vs Libbloom vs BionetBF): Comparison of performance among Cuckoo Filter, Libbloom, and BionetBF based on Megabyte per second for query operations in case of (a) Same Set, (b) Mixed Set, and (c) Disjoint Set. Higher is better. 10M: 10 million, 20M: 20 million, 30M:million, and 40M: 40 million number of IDKmers queried.

approximately 1.704×, 1.42×, and 1.13× (on average) better than Libbloom in Same Set, Mixed Set and Disjoint Set, respectively. Moreover, Table 5 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(25) (Code available at (26)). 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.

BionetBF with Biological network

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

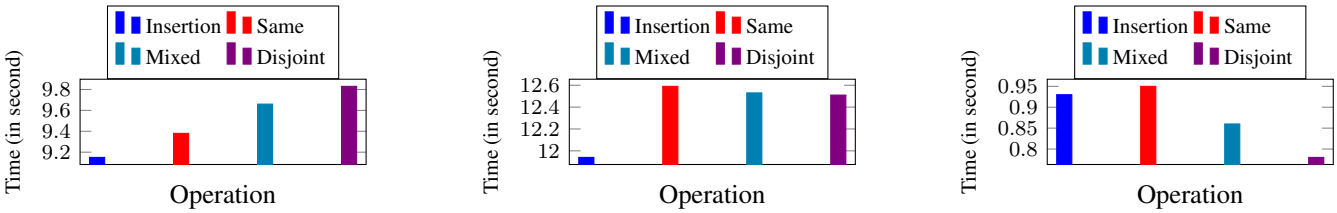
Table 6. BPS of biological network

This section experimented 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 . 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, 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.

Fig. 28 represents the insertion and query time of BionetBF with the bionetwork dataset. Chemical-Gene, Gene-Disease, and Gene-Gene takes 9.15 secs, 11.937 secs and 0.925 secs 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 secs, 12.51 secs, and 0.775 secs respectively. Fig 29 illustrates the FPP of BionetBF with bionetwork datasets: Chemical-Gene (Figure 29a) and Gene-Gene (29a). Gene-Gene dataset has zero FPP. Both Chemical-Gene and Gene-Disease dataset are of big sized dataset, however, the FPP is very low. Fig 30 illuminates the accuracy of BionetBF with bionetwork datasets: Chemical-Gene (Figure 30a), Gene-Disease (Figure 30b) and Gene-Gene (30c). Both Chemical-Gene and Gene-Gene dataset have 100% accuracy; but Gene-Disease has accuracy more than 99.9%. Fig 31 elucidate the performance of BionetBF with the bionetwork datasets for both insertion and 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.

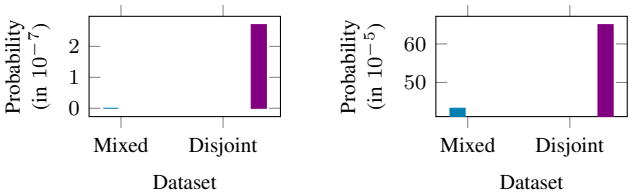
DISCUSSION

The knowledge of biological network is very important in drug discovery. The 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



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

Figure 28. 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

Figure 29. 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.

biological network with thousands or millions of nodes are stored in a small sized BionetBF. 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. Let's suppose a pharmaceutical company wants to study the compatibility 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 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.

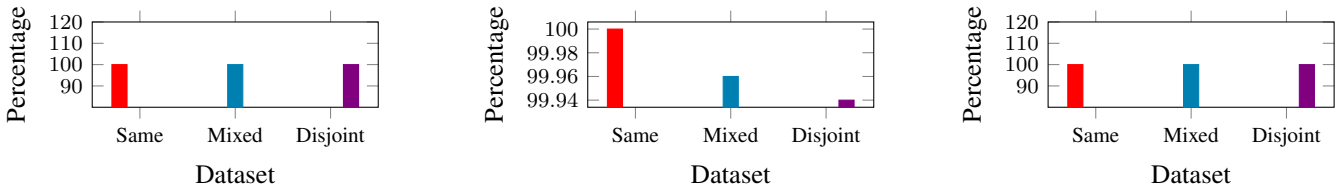
Bloom Filters are mostly known for deduplication. BionetBF is also an excellent choice for deduplication of repetitive insertion or query of paired biological data. While determining the biological network of a biological process, the same paired biological data is generated and forwarded to the database 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.

BionetBF has a limitation too. The size of the filter (i.e., the data structure) of BionetBF is fixed. Hence, if BionetBF becomes saturated, the FPP increases, but reconstruction of the BionetBF requires insertion of the sequences from the start. BionetBF is not a database; it cannot provide the list of inserted sequences. Also, it is not possible to retrieve one biological data of the paired biological data from another biological data. False positives are an issue of all Bloom Filters; BionetBF has less FPP; however, it is not zero in all cases.

CONCLUSION

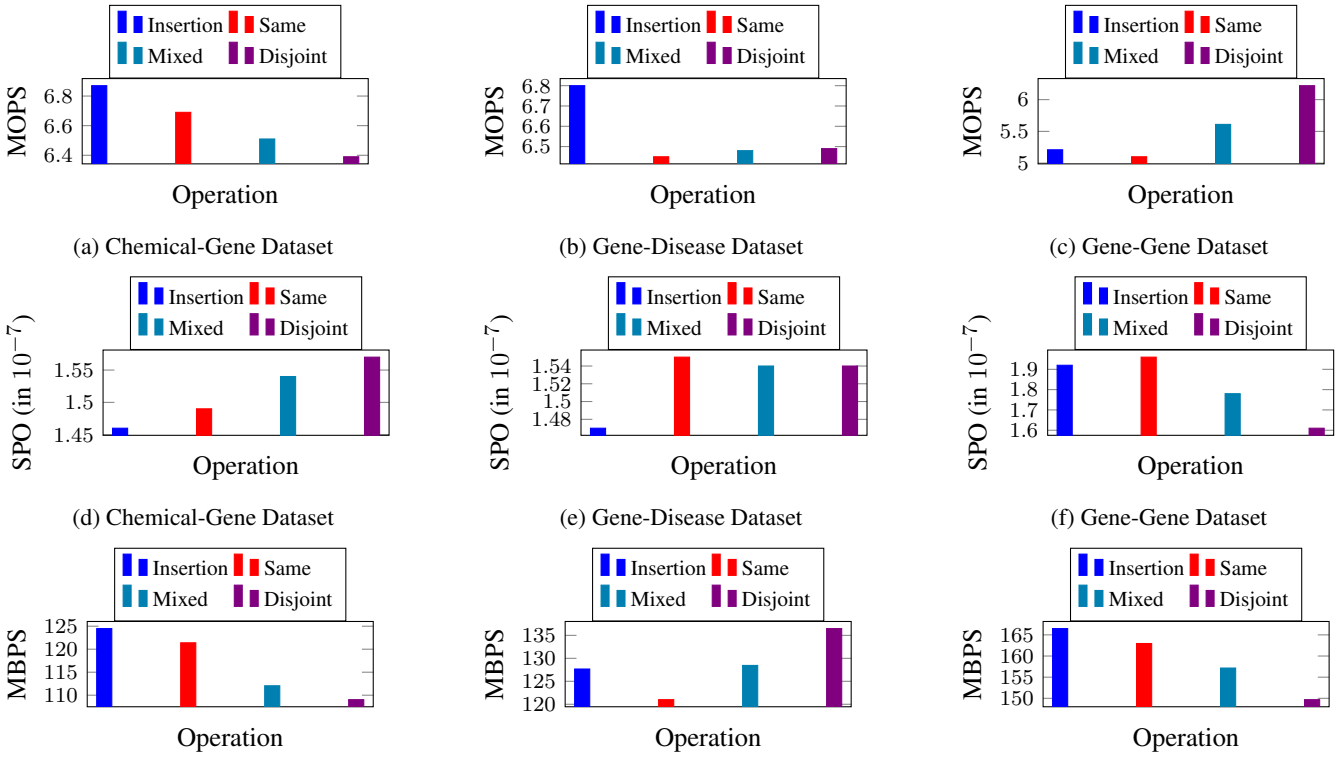
In this article, we have presented a novel Bloom Filter, called BionetBF, for membership identification of paired biological data. BionetBF, is a small filter size that stores a huge amount of biological data. We have conducted a series of experiments using 12 generated datasets and three biological network datasets. 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

cture(0,0)(-30,0)10 (-30,-5)(0,1)10 (-35,0)(1,0)30 (0,30)10 (-5,30)(1,0)10 (0,35)(0,-1)30 picturepicture(0,0)(30,0)10 (30,-5)(0,1)10 (35,0)(-1,0)30 (0,30)10 (-5,30)(1,0)10 (0,35)(0,-1)30 picture



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

Figure 30. 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

Figure 31. 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.

filters. Therefore, BionetBF can be used the enhance the entire system performance of Drug Discovery, Genome coding, etc., due to its low memory footprint, accuracy and performances.

CODE AVAILABILITY

The source code is available at `https://github.com/patgiri/BionetBF`. The code is written in C programming language.

Conflict of interest statement. The authors declare that there is no conflict of interest to disclose.

REFERENCES

1. Badaloni, S., Camillo, B. D., and Sambo, F. (2012) Qualitative Reasoning for Biological Network Inference from Systematic Perturbation Experiments. IEEE/ACM Transactions on Computational Biology and Bioinformatics, **9**(5), 1482–1491.
2. Di Camillo, B., Falda, M., Toffolo, G., and Cobelli, C. (2011) SimBioNeT: a simulator of biological network topology. IEEE/ACM transactions on computational biology and bioinformatics, **9**(2), 592–600.
3. Boucher, B. and Jenna, S. (2013) Genetic interaction networks: better understand to better predict. Frontiers in Genetics, **4**, 290.
4. Lecca, P. and Priami, C. (2013) Biological network inference for drug discovery. Drug Discovery Today, **18**(5), 256–264.
5. Kuzmanov, U. and Emili, A. (2013) Protein-protein interaction networks: probing disease mechanisms using model systems. Genome medicine, **5**(4), 1–12.
6. Mulder, N. J., Akinola, R. O., Mazandu, G. K., and Rapanoel, H. (2014) Using biological networks to improve our understanding of infectious diseases. Computational and Structural Biotechnology Journal, **11**(18),

1–10.
7. Ogata, H., Fujibuchi, W., Goto, S., and Kanehisa, M. (2000) A heuristic graph comparison algorithm and its application to detect functionally related enzyme clusters. Nucleic acids research, **28**(20), 4021–4028.
8. Moreno, L. Q. (2019) Graphlets and Motifs in Biological Networks. In Ranganathan, S., Gribskov, M., Nakai, K., and SchÃ¶nbach, C., (eds.), Encyclopedia of Bioinformatics and Computational Biology, pp. 814–820 Academic Press Oxford.
9. Zhang, B., Tian, Y., and Zhang, Z. (2014) Network biology in medicine and beyond. Circulation: Cardiovascular Genetics, **7**(4), 536–547.
10. Salem, M. S. Z. (2018) Biological networks: An introductory review. Journal of Proteomics and Genomics Research, **2**(1), 41–111.
11. Nayak, S., Patgiri, R., and Borah, A. (2021) A survey on the roles of Bloom Filter in implementation of the Named Data Networking. Computer Networks, **196**, 108232.
12. Bloom, B. H. (1970) Space/time trade-offs in hash coding with allowable errors. Comm. of the ACM, **13**(7), 422–426.
13. Appleby, A. Murmur Hashing. Retrieved from https://sites.google.com/site/murmurhash/ (2010).
14. Patgiri, R., Nayak, S., and Borgohain, S. K. (2020) PassDB: A password database with strict privacy protocol using 3D Bloom filter. Information Sciences, **539**, 157–176.
15. Patgiri, R., Nayak, S., and Muppalaneni, N. B. (2021) Is Bloom Filter a Bad Choice for Security and Privacy?. In 2021 International Conference on Information Networking (ICOIN) pp. 648–653.
16. Fan, B., Andersen, D. G., Kaminsky, M., and Mitzenmacher, M. D. (2014) Cuckoo Filter: Practically Better Than Bloom. In Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies New York, NY, USA: Association for Computing Machinery CoNEXT '14 p. 75–88.
17. Pagh, R. and Rodler, F. F. (2004) Cuckoo hashing. Journal of Algorithms, **51**(2), 122–144.
18. Patgiri, R., Nayak, S., and Borgohain, S. K. (2019) rDBF: A r-Dimensional Bloom Filter for massive scale membership query. Journal of Network and Computer Applications, **136**, 100–113.
19. UCSC Genome Browser Downloads. `http://hgdownload.cse.ucsc.edu/downloads.html` (Accessed on 15/03/2021).
20. Mambo: Multimodal Biomedical Networks. `http://snap.stanford.edu/mambo/` (Accessed on 02/04/2021).
21. Goh, K.-I., Cusick, M. E., Valle, D., Childs, B., Vidal, M., and Barabási, A.-L. (2007) The human disease network. Proceedings of the National Academy of Sciences, **104**(21), 8685–8690.
22. Gilbert-Diamond, D. and Moore, J. H. (2011) Analysis of gene-gene interactions. Current protocols in human genetics, **70**(1), 1–14.
23. GitHub - efficient/cuckoofilter. `https://github.com/efficient/cuckoofilter` (Accessed on 10/05/2021).
24. GitHub - jvirkki/libbloom: A simple and small bloom filter implementation in plain C.. `https://github.com/jvirkki/libbloom` (Accessed on 21/05/2021).
25. Graf, T. M. and Lemire, D. (March, 2020) Xor Filters: Faster and Smaller Than Bloom and Cuckoo Filters. ACM J. Exp. Algorithmics, **25**.
26. GitHub - FastFilter/xor_singleheader: Header-only Xor Filter library. `https://github.com/FastFilter/xor_singleheader` (Accessed on 02/05/2021).