FANGS: High Speed Sequence Mapping for Next generation Sequencers

Sanchit Misra
Electrical Engineering and Computer Science
Northwestern University
Evanston, IL 60208

Ramanathan Narayanan
Electrical Engineering and Computer Science
Northwestern University
Evanston, IL 60208

Simon Lin
Robert H. Lurie Comprehensive Cancer Center
Northwestern University
Chicago, IL 60611

Alok Choudhary
Electrical Engineering and Computer Science
Northwestern University
Evanston, IL 60208

ABSTRACT

Next Generation Sequencing machines are generating millions of short DNA sequences (reads) everyday. There is a need for efficient algorithms to map these sequences to the reference genome to identify SNPs or rare transcripts and to fulfill the dream of personalized medicine. We present a **F**ast **A**lgorithm for **N**ext **G**eneration **S**equencers (FANGS), which dynamically reduces the search space by using q-gram filtering and pigeon hole principle to rapidly map 454-Roche reads onto a reference genome. FANGS is a sequential algorithm designed to find all the matches of a query sequence in the reference genome tolerating a large number of mismatches or insertions/deletions. Using FANGS, we mapped 50000 reads with a total of 25 million nucleotides to the human genome in as little as 23.3 minutes on a typical desktop computer. Through our experiments, we found that FANGS is upto an order of magnitude faster than the state-of-the-art techniques for queries of length 500 allowing 5 mismatches or insertion/deletions.

Keywords

Sequence Mapping, Next Generation Sequencers, 454 Sequencers

1. INTRODUCTION

Recent advances in Next Generation Sequencing (NGS) technology have led to affordable desktop-sized sequencers with low running costs and high throughput. These sequencers produce small fragments of the genome being sequenced as a result of the sequencing process. For example, the Roche-454 system can generate $400,000$ sequences of length 250-500 nucleotides in a 7.5 hour run [12]. The Illumina-Solexa system generates smaller but many more sequences. It can generate 50 million sequences of length 30-50nt in just 3 days [5]. The ABI-SOLiD system can also generate data at a similar rate [5]. This is a rapidly advancing field with a very high rate of increase in throughput. It is speculated that the running costs of sequencing a genome will eventually be as low as \$1000 [16]. This major technological breakthrough allows next generation sequencing to be used in a variety of biological analyses, including metagenomics, SNP discovery, comparative genomics, gene expression, genotyping and personal genomics.

A key step in many of these applications is to map these short sequences, called reads, to a reference genome, to find the locations where each read occurs in the genome, allowing for a small number of mismatches or insertion/deletions. Typically, the length of the query sequence can range from $30-50$ (generated by Illumina-Solexa sequencers) to 250-500 (generated by Roche-454 sequencers). A typical genomic database, for instance, the human genome, can be 3 billion nucleotides in length. In this article, we focus on the mapping of Roche-454 system reads. The longer length reads produced by 454-sequencers ensure less ambiguity in mappings. A 454 sequencer was recently used for sequencing the DNA sequence of James D. Watson to 7.4 fold redundancy in just two months [19]. The authors used BLAT [6] to map the 454 reads to a reference genome, which is not at par with the sequencing speed. Moreover, BLAT is designed for local alignment, while sequence mapping requires the entire length of a query to be mapped. There have been considerable efforts to develop faster sequence mapping tools which can match the speed of Next Generation Sequencers, but most of them have been for reads generated by Illumina-Solexa machines (for example, ELAND, MAQ [8], SOAP [9] and BowTie [7]). Even though 454 sequencers are widely used by researchers, there has not been sufficient research to develop faster tools for mapping 454 reads. To the best of our knowledge, the only algorithm which is specifically designed for 454 data is BWA, which is an unpublished package written by the authors of Maq. BWA is based on Burrows-Wheeler Transform (BWT). It supports gapped global alignment with respect to queries and is one of the fastest short read alignment algorithms while also finding suboptimal matches. But, we demostrate in our results that it suffers from low sensitivity. Therefore, there is a need to design powerful algorithms and systems which can efficiently and accurately map 454 reads.

In this article, we describe our algorithm FANGS, a Fast Algorithm for Next Generation Sequencers, which dynamically reduces the search space by using q-gram filtering and the pigeonhole principle, to rapidly map 454 reads onto a reference genome. FANGS is a sequential algorithm designed to find all the matches of a query sequence in the reference genome tolerating a large number of mismatches or insertions/deletions. It uses an efficient lossy data structure which requires just 1GB memory to store the index of the human genome, which is 3 giga-bases in size and still achieves nearly 100% sensitivity. FANGS supports FASTA input file format and many output file formats and comes with command line options for controlling almost every aspect of the mapping process. In comparison with existing tools, the most significant features of FANGS are:

- High flexibility. It allows a large number of mismatches and insertions/deletions in mapping.
- High Sensitivity. It tries to find all the matches for each query and maps nearly 100% of the queries.
- Ability to handle large datasets. Using FANGS, we have mapped $50,000$ queries of length 500 each to a human genome in as little as 23.3 minutes.
- Speed. FANGS is upto an order of magnitude faster than the state-of-the-art techniques for queries of length 500 allowing 5 mismatches or insertion/deletions.

The beta version of FANGS is being managed and freely available at http://www.ece.northwestern.edu/∼smi539/fangs.html.

The remainder of the paper is organized as follows. We give a formal definition of the problem in Section 2 followed by related work in Section 3. Section 4 describes our algorithm in detail followed by results in Section 5 and conclusion in Section 6.

2. THE SEQUENCE MAPPING PROBLEM

In the context of Next Generation Sequencing, sequence mapping problem involves searching for a small DNA sequence (read) in the reference genome allowing a small number of differences. The reference genome is obtained from an organism of the same species as the reads, implying a high level of similarity between the read and the reference genome. The small number of differences are allowed to account for differences between individual organisms and sequencing errors.

Given a string S over a finite alphabet Σ, we use $|S|$ to denote the length of S, $S[i]$ to refer to the i^{th} character of S and $S[i:j]$ to refer to the substring of S which starts at position i and ends at position j. A *q-gram* of S is a substring of S of length $q > 0$. The unit cost edit distance between two strings S_1 and S_2 is the minimum number of substitutions, insertions and deletions required to convert S_1 to S_2 [15]. We will refer to the unit cost edit distance between S_1 and S_2 as $edist(S_1, S_2)$. It can be calculated by using Needleman-Wunsch algorithm in $O(|S_1||S_2|)$ time [10]. For a string S, we will use $dec(S, \Sigma)$ to refer to the natural decimal representation of S over Σ. For example, for $\Sigma = \{A, C, G, T\}$, the nucleotides A, C, G, T are mapped to the numbers $0, 1, 2, 3$ respectively. Therefore:

$$f(A) = 0, f(C) = 1, f(G) = 2, f(T) = 3,$$

And, $dec(S, \{A, C, G, T\}) = \sum_{i=0}^{|S|-1} 4^i f(S[i])$

This brings us to the formal definition of the short read sequence mapping problem. Every genomic sequence can be represented as a string over the alphabet $\Sigma = \{A, C, G, T\}$. Given a genome database G of subject sequences $\{S_1, S_2, \cdots, S_m\}$, a query sequence (read) Q of length l and an integer n, it is required to find all substrings from G, such that for each substring α, $edist(\alpha, Q) < n$. We will refer to the integer n as the *maxEditDist* parameter. There have been studies to find the best match out of all matches satisfying the edit distance criteria. However, since the edit distance can be due to a number of reasons, including sequencing errors, there is no definite way of knowing which one is the best. Hence, we are focussing on finding all the matches with edit distance less than *maxEditDist*.

3. RELATED WORK

The classical approach to sequence alignment involves several variants of dynamic programming, the most prominent being the algorithms of Needleman-Wunsch [10] and Smith-Waterman [18]. Dynamic programming is prohibitively expensive in terms of time and space for larger sequences like the human genome, and this has led to the development of faster hash-table based heuristic methods like FASTA [13], BLAST [4], BLAT [6] and SSAHA [11]. BLAST has been the most popular tool for sequence alignment since its creation in 1990. However, it usually takes several hundreds of days for the data generated in just a few hours by the latest sequencers and hence is not a practical solution. Recently, the advent of Next Generation Sequencers has prompted researchers to develop high-performance sequence mapping tools. Some of the most prominent tools for sequence mapping include ELAND, MAQ [8], RMAP [17], SOAP [9], SWIFT [15], SHRiMP [1], SeqMap [5], BowTie [7], GMAP [20], Mosaik [2], BWA [3] and SSAHA2 [11]. The key idea behind these algorithms is the following lemma from [14].

Lemma 1: If two strings $A[1..m]$ and $B[1..m]$ have at most n mismatches and $p = \lfloor \frac{m}{n+1} \rfloor$, then there must be an integer i such that $A[i : i+p-1] = B[i : i+p-1]$. In other words, A and B share a common substring of length p.

The following corollary can be easily derived from Lemma 1.

Corollary 1: Given a genome $G[1..L]$ and a query $Q[1..m]$ $(L > m)$, if there is a substring α of G, such that α and Q match with an edit distance of at most n and $p = \lfloor \frac{m}{n+1} \rfloor$, then there must exist i, j such that $G[i : i+p-1] = Q[j : j+p-1]$.

The substring α is called a homologous region of Q in G. Most sequence mapping algorithms first find the locations of all the candidate homologous regions of G which can potentially have an edit distance of less than the *maxEditDist* by using a criteria similar to Corollary 1. These candidate regions are then checked using an accurate algorithm to see if the edit distance is indeed less than *maxEditDist*.

The closest precursor of our sequential algorithm is the program SWIFT [15]. A *q-hit* between two strings S_1 and

```
Algorithm $GetHits(seq, dbIndex, q)$
Input:
$seq$ : the query sequence.
$dbIndex$ : The index-table of the database obtained after preprocessing
$q$ : q-gram size used for the creation of q-gram index.
Output:
$hitList$ : List of all q-hits.
$wildCardList$ : List of query indices for which the q-gram starting at that index is more frequent than maxFreq
 1: $hitList \Leftarrow \phi$
 2: $wildCardList \Leftarrow \phi$
 3: for $i$ in 0 to $len(seq) - q + 1$ do
 4:   $locations$, $numLocations \Leftarrow getDBLocations(\ dbIndex,\ seq[i : i + q - 1])$
 5:   if $numLocations = -1$ then
 6:     $addToList(wildCardList, i)$
 7:   else
 8:     for $j$ in 1 to $numLocations$ do
 9:       $hit.dStart = locations[j]$
10:       $hit.qStart = i$
11:       $addToList(hitList, hit)$
```

Figure 1: Algorithm for obtaining *q-hits*

S_2 is the tuple (i, j) such that $S_1[i : i+q-1] = S_2[j : j+q-1]$. SWIFT creates an index of all *q-grams* in the database, called *q-gram index*, and uses it to find all *q-hits* of G and Q. It then identifies regions that have a certain minimum number of hits. These regions are further analyzed to check if the edit distance is within limits. FANGS also creates an index of *q-grams* in the database. However, it does not include highly frequent *q-grams* in the index thereby saving time and memory. It compensates for the data loss by using a wildcard strategy, which is described in the following section.

4. FANGS: FAST ALGORITHM FOR NEXT GENERATION SEQUENCERS

4.1 Preprocessing step: Creation of the q-gram index

Here we describe the construction of the *q-gram index*. We preprocess the sequences in the database by breaking them into non-overlapping *q-grams* and store the location of each *q-gram* in the *q-gram index*. We will refer to the *q-gram index* as the *index-table*. We refer to the size of these non-overlapping *q-grams*, q, as *tileSize*. Each *q-gram* t can be uniquely mapped to a corresponding integer $dec(t, \Sigma)$ as defined in Section 2.

For each *q-gram* t, we calculate two values: (1) $tileHead(t) = dec(t[1 : 12], \Sigma)$ and (2) $tileTail(t) = dec(t[13 : q], \Sigma)$. The *index-table* consists of two arrays. The first array *occurrenceTable* stores (i) the location of $t[1 : 12]$ in the database G and (ii) $tileTail(t)$ for each *q-gram*. Hence, *occurrenceTable* contains the concatenation of lists $L(t[1 : 12]) = \{i, tileTail(G[i : i + q - 1]) | G[i : i + 11] = t[1 : 12]\}$, where t is a *q-gram*, that is $t \in \Sigma^q$. For each *q-gram* $t \in \Sigma^q$, the position $tileHead(t)$ in the second array *lookupTable* contains the pointer $p(t)$, which points to the beginning of the correponding list $L(t[1 : 12])$ in the *occurrenceTable*; and the count $c(t)$ of the number of occurences of $t[1 : 12]$ in G. Hence the length of the *lookupTable* is $|\Sigma|^{12}$. In order to find hits for a *q-gram* t, it first indexes the *lookupTable* with $tileHead(t)$. Let $L(t[1 : 12])$ be the corresponding list. The *q-hits* can be found by traversing through the list and outputting those locations for which $tileTail(t)$ matches.

The *index-table* is created in two passes. In the first pass, we find the number of non-overlapping occurrences of each *q-gram* in the database, so that we can allocate appropriate amount of memory to the *occurrenceTable* and calculate the pointer positions for *lookupTable*. In the second pass, we fill the array *occurrenceTable* with appropriate values for the *q-grams*. Note that creation of index need to be done only once for a given value of q. After that, we can process any number of queries.

The above structure of q-gram index is similar to the one used by BLAT [6]. The main difference is that BLAT stores the locations of all the q-grams for $q > 12$. But this greatly reduces the speed of the algorithm. We ignore all the *q-grams* with number of occurrences in the database greater than a certain threshold frequency, say *maxFreq*. Hence for such highly frequent *q-grams*, FANGS makes the locations corresponding to frequently occurring *q-grams* in the *occurrenceTable* as -1. This filtering step helps us in two ways. First, it reduces the index table size. Second, it avoids unnecessary false hits due to repetition of DNA thereby improving efficiency. Due to this technique, we need only 1GB memory to store the index of the human genome that is 3GB in size.

4.2 Using index-table to map sequences

A substring of size p has at least $\lfloor \frac{p-(q-1)}{q} \rfloor$ *q-grams*. Substituting p from corollary 1, we get:

Corollary 2: Given a query $Q[1..m]$ and database $G[1..L]$ ($m < L$). For all substrings α of G such that $edist(Q, \alpha) < n$, $\exists i, j$ such that $Q[i : i + q - 1] = \alpha[j : j + q - 1], Q[i + q : i + 2q - 1] = \alpha[j + q : j + 2q - 1], \cdots, Q[i + (T - 1)q : i + Tq - 1] = \alpha[j + (T - 1)q : j + Tq - 1]$, where T is given by:

$$T = \lfloor \frac{\lfloor \frac{m}{n+1} \rfloor - (q - 1)}{q} \rfloor$$

We use the above corollary to dynamically reduce the

Algorithm $FindRegions(hitList, wildCardList, q, size, \text{n}, T(>1))$
Input:
hitList : List of all *q-hits* $<d_j, r_k>$, where d_j is the *dStart* value and r_k is the *qStart* value.
wildCardList : List of query indices for which the *q-gram* starting at that index is more frequent than *maxFreq*
q : *q-gram* size used for the creation of *index-table*.
size : size of the query sequence.
n : maximum edit distance allowed in the mapping (*maxEditDist*).
T : T as given in corollary 2
Output:
regionList : list of candidate homologous regions.

```
 1: $sortList(hitList, dStart)$
 2: Let $d_1, d_2, ..., d_x$ be the distinct dStart values in ascending order.
 3: for all $i$ do
 4:    Let $< d_i, r_{i,1} >,< d_i, r_{i,2} >, \cdots, < d_i, r_{i,y} >$ be the hits containing $d_i$
 5:    Let $< d_{i+1}, r_{i+1,1} >,< d_{i+1}, r_{i+1,2} >, \cdots, < d_{i+1}, r_{i+1,z} >$ be the hits containing $d_{i+1}$
 6:    if there exists $a$, $b$ such that $r_{i+1,b} - r_{i,a} = d_{i+1} - d_i$ then
 7:       if $d_{i+1}$ and $d_i$ are either adjacent in the database or are separated only by highly frequent q-grams then
 8:          add $< d_{i+1}, r_{i+1,b} >$ to the matchingBlock containing $< d_i, r_{i,a} >$
 9:          if size of matchingBlock $\geq T$ then
10:             add the matchingBlock to the matchingBlockList
11: $regionList \Leftarrow \phi$
12: for all blockHit in matchingBlockList do
13:    $region.dBegin \Leftarrow blockHit.dStart - blockHit.qStart - n$
14:    $region.dEnd \Leftarrow blockHit.dStart - blockHit.qStart + size - 1 + n$
15:    $addToRegionList(regionList, region)$
```

Figure 2: Algorithm for stitching together q-hits to find candidate homologous regions

search space and map query sequences at a very high speed. For each query, we first find the list of candidate homologous regions. The algorithm for finding the candidate homologous regions can be divided in three steps. First we find all the *q-hits* of the query in the database. Each *q-hit* consists of two values : starting position of the *q-gram* in the query (*qStart*) and in the database (*dStart*). The algorithm is given in Figure 1. The algorithm takes each overlapping *q-gram* in the query and finds the locations of all occurrence of the *q-gram* in the database using the *index-table*. The algorithm creates a hit with each location and adds it to the hitList. For some of the *q-grams* which are more frequent than the *maxFreq* parameter, the number of locations *numLocations* is returned as -1. We add all such query indices to the *wildCardList*.

In the next step, we stitch together *q-hits* which are adjacent to each other both in the query and the database to create maximal *matchingBlocks*. A *block* is defined as a contiguous sequence of *q-grams* in a sequence. Also, we define a *matchingBlock* as a *block* in the query that perfectly matches a *block* of same length in the database. We represent a *matchingBlock* as a tuple $(qStart, dStart, len)$, where qStart and *dStart* are the starting locations of the first *q-gram* of the *matchingBlock* in the query and the database respectively and *len* is the number of *q-grams* in the *matchingBlock*. A maximal *matchingBlock* is a *matchingBlock* which will result in a mismatch if extended any further on either side. The algorithm is given in Figure 2.

As the hits are obtained in the order of increasing *qStart* values, the hitList is already sorted according to the *qStart* values. Now we sort the list according to database positions (*dStart* values). As a result, the list is now sorted according to the *dStart* values and for each *dStart* value, it is sorted according to the *qStart* values. Let $d_1, d_2, ..., d_x$ be the distinct *dStart* values in ascending order. For each i, we check if $d_{i+1} - d_i$ is equal to q; i.e.; they are the neighboring *q-grams* in the database. Let $< d_i, r_{i,1} >,< d_i, r_{i,2} >, \cdots, < d_i, r_{i,y} >$ be the hits containing d_i and $< d_{i+1}, r_{i+1,1} >,< d_{i+1}, r_{i+1,2} >, \cdots, < d_{i+1}, r_{i+1,z} >$ be the hits containing d_{i+1}. If $d_{i+1} - d_i = q$, then we search for a pair of hits $< d_i, r_{i,a} >$ and $< d_{i+1}, r_{i+1,b} >$ such that $r_{i+1,b} - r_{i,a} = q$. This means the pair of hits has neighboring tuples both in the query and the database. Hence, we have a *matchingBlock* of length 2 *q-grams*. This way we keep on combining hits to form *matchingBlocks*. If the length of a *matchingBlock* $\geq T$, we store it in the *matchingBlockList*.

This technique does not capture all the *matchingBlocks* with length greater than or equal to T because we do not store the database locations of very frequently occuring *q-grams*. Since multiple *matchingBlocks* may contain the frequently occuring *q-grams*, this leads to some of them not being detected. In order to solve this problem, we give a wildcard to all the frequently occuring *q-grams*. According to this wildcard, the frequently occuring *q-grams* can be part of any *matchingBlock*. Therefore, if two *matchingBlocks* are separated only by frequently occuring *q-grams*, the two of them together with the frequently occuring *q-grams* are combined to form one big *matchingBlock*.

Once we have the *matchingBlockList*, we extend each *matchingBlock* in the list to create a candidate homologous region. We also keep a buffer of size n on either side to account for gaps in the alignment. Each region consists of two values - beginning location, *dBegin* and end location, *dEnd* of the region in the database. Hence the beginning of the *matchingBlock* would be:

$$dBegin = dStart - qStart - n$$

Algorithm $FANGS(seqList,\ seqCount,\ db,\ dbIndex,\ n,\ q,\ outFile)$
INPUT:
seqList : list of query sequences.
seqCount : total number of query sequences
db : genomic database.
dbIndex : The *index-table* of the database obtained after preprocessing
n : maximum edit distance allowed in the mapping (*maxEditDist*).
q : *q-gram* size used for the creation of *index-table*.
outFile : file where output has to be written.
Output:
All mappings of each query sequence in *seqList* in the database *db*

```
1: for all $seq$ in $seqList$ do
2:   $hitList \Leftarrow GetHits(seq, dbIndex, q)$
3:   $T = \lfloor \frac{\lfloor \frac{m}{n+1} \rfloor - (q-1)}{q} \rfloor$
4:   $regionList \Leftarrow FindRegions(hitList, wildCardList, q, size, n, T)$
5:   for all $region$ in $regionList$ do
6:     if $edist(region, seq) \leq n$ then
7:       $outputmapping(region, seq)$
```

Figure 3: The sequence mapping algorithm

and the end would be:

$$dEnd = dStart - qStart + size - 1 + n$$

Thus the homologous region is created by extending the *matchingBlock* on either side to cover the whole query and adding a buffer of n bases on either side. The function *addToRegionList* ensures that we do not add two regions which have a huge overlap as they will result in the same mapping. If one region completely covers another region, we only include the former. Moreover, if two regions have an overlap of more than a certain value, then we merge them together into one region. This is done to avoid multiple outputs for the same homologous region. The potential homologous region is further processed by using an adaptation of the Needleman-Wunsch algorithm to check if the homologous region actually has an edit distance $\leq n$. Since we are trying to find regions with a maximum edit distance of n, we only need to calculate the diagonal band of width $2n+1$ of the matrix used for Needleman-Wunsch algorithm. The complete algorithm is as given in Figure 3.

The novelty of FANGS lies in the fact that it stitches the hits obtained into contiguous blocks of query which exactly match a contiguous block in the database. Another important contribution is that it gives a wildcard to all highly frequent *q-grams*. Hence, even though we do not store the highly occurring *q-grams* in the *index-table*, we can still map queries with 100% sensitivity.

5. RESULTS AND DISCUSSION

In this section, we describe the various results obtained by running FANGS on different datasets. We also compare our results with those of other state-of-the-art tools. The focus of this article is 454 sequencing data. Hence, we only consider query lengths within the range of $300 - 500$ in our experiments.

5.1 Dataset description and experimental setup

We performed our experiments on reads from the 1000 Genomes project pilot (National Center for Biotechnology Information [NCBI] Short Read Archive : SRR005010, SRR005011, SRR005012, SRR005013). These reads are obtained from the LS454 platform and are part of the SRX001297 experiment. We created the following three pools of queries using the short read archive:

- SRX001297_300 : Obtained by filtering reads smaller than 300 in length and trimming the resulting reads from right to a length of 300.
- SRX001297_400 : Obtained by filtering reads smaller than 400 in length and trimming the resulting reads from right to a length of 400.
- SRX001297_500 : Obtained by filtering reads smaller than 500 in length and trimming the resulting reads from right to a length of 500.

We ran FANGS for all these different sets of queries. We also evaluated the performance of several currently existing sequence alignment tools to compare our performance. For all our experiments, our database consists of the hg19 version of unmasked human genome. For our evaluation, we ran all the experiments on Intel Xeon quad core E5430 2.66 GHz processor with 2x6MB cache and 32GB RAM running a Linux based operating system. In our experiments, none of the sequence mapping tools considered the quality information.

5.2 Comparison with BWA, Mosaik and BLAT

Table 1 compares FANGS with BWA, Mosaik and BLAT [6]. BWA is a mapping tool for 454 reads developed by Heng Li, the author of MAQ [8]. For this comparison, we used the real reads obtained from 1000 genomes project. The query sets were created by randomly selecting 50000 queries each from SRX001297_300, SRX001297_400 and SRX001297_500. All tools were run so as to allow a maximum edit distance of 5. As mentioned earlier, with FANGS we are focussing on finding all possible mappings of reads produced by 454 sequencers. We ran BWA with the -N option so that it outputs all possible alignments. The table shows that FANGS produces more alignments at a much faster speed. For Mosaik,

Table 1: Mapping results of FANGS, BWA(bwa-0.4.6), Mosaik(Mosaik-0.9.891) and BLAT (version 34) for 50000 queries of length 300, 400 and 500 against the human genome hg19 allowing a maximum edit distance of 5. Third column specifies the sum of the number of mappings found for each query.

Query length	Program	Number of mappings found	Time taken (min)	Peak virtual memory footprint (gigabytes)	Percentage of reads mapped	Reads mapped per hour
300	FANGS	96230	37.0	4.5	92.00	81195
	BWA	36046	177.1	2.3	72.09	16942
	Mosaik	44433	109.4	1.8	87.79	27416
	BLAT	83123	97.9	3.8	85.52	30635
400	FANGS	57256	26.6	4.5	86.27	112627
	BWA	27248	128.3	2.3	54.50	23385
	Mosaik	41546	182.8	1.8	82.48	16410
	BLAT	51884	176.6	3.8	75.10	16985
500	FANGS	41014	23.3	4.5	73.23	128893
	BWA	15693	67.8	2.3	31.39	44227
	Mosaik	34877	278.8	1.8	69.45	10760
	BLAT	32586	250.1	3.8	57.14	11994

Table 2: This table shows a matrix of similarity in mapping between various tools. Each row and column of the matrix is marked by a mapping tool. Each entry in the matrix gives the percentage of queries mapped by the row tool that are also mapped by the column tool. For example, FANGS maps 99.98% of queries mapped by Mosaik for query length 300.

Query length	Mapping similarity				
300		FANGS	Mosaik	BWA	BLAT
	FANGS	100	95.40	78.35	92.33
	Mosaik	99.98	100	78.66	92.60
	BWA	99.99	95.79	100	99.32
	BLAT	99.33	95.06	83.73	100
400		FANGS	Mosaik	BWA	BLAT
	FANGS	100	95.57	63.16	86.40
	Mosaik	99.96	100	63.30	86.62
	BWA	99.99	95.82	100	98.96
	BLAT	99.26	95.14	71.81	100
500		FANGS	Mosaik	BWA	BLAT
	FANGS	100	94.83	42.86	77.44
	Mosaik	99.98	100	42.91	77.84
	BWA	99.99	94.97	100	98.41
	BLAT	99.25	94.62	54.06	100

we ran it for each chromosome separately since it needed more than 32GB memory to map queries against all the chromosomes at once. Since BLAT does not have any way of specifying the maximum allowed edit distance, we used the minScore parameter to approximately simulate them. If we run BWA without the -N option, it runs much faster (takes 13.8min, 12.7min and 10.4min for 50000 queries of length 300, 400 and 500 respectively) but produces only one mapping per query. We also tried to compare with Seqmap [5], but the gapped alignment did not finish after running for more than two days.

Table 2 shows similarilty in the queries mapped by each tool. Note that, FANGS maps more than 99.25% of the queries mapped by Mosaik, BLAT and BWA. Moreover, Mosaik maps approximately 95% of the queries mapped by FANGS. This shows a high degree of similarity between the results of the two tools. It also shows that FANGS maps approximately all the queries that Mosaik, BLAT and BWA map.

5.3 Comparison with Bowtie, SOAP, MAQ and SHRiMP

In this section, we compare FANGS against leading mapping tools for Illumina/Solexa data. We understand that these tools are specifically designed for shorter reads and it might not be an appropriate comparison. The query sets were created by randomly selecting 50000 queries each from SRX001297_300, SRX001297_400 and SRX001297_500. MAQ does not allow queries of length > 128 and SHRiMP did not finish in more than two days. So, we could not compare with them. Bowtie and SOAP do not allow more than 2 mismatches. Hence, for a fair comparison, we ran FANGS with a maximum edit distance of 2. Table 3 compares the results. FANGS is significantly slower than both Bowtie and SOAP but finds many more alignments and maps many more queries. Moreover, allowing 2 mismatches is not sufficient for 454 reads. Hence, as expected, tools specifically designed for Illumina/Solexa data can not be used for mapping 454 reads and hence there is a need of tools specifically designed for longer 454 reads.

5.4 Experiments with synthetic reads

Although it is better to perform experiments with real data, it is impossible to assess the accuracy of read mapping, because we cannot know where the reads came from and how many errors are there in each read. Hence, in addition to real reads, we also created some simulated reads. In order to create reads, we took the whole human genome and filtered it to get a set of regions in the genome which were free of N's. In order to get queries of length l, we sampled each of these regions for strings of length l. We randomly changed 5 bases in each query to introduce 5 SNPs. This gave us 9537575,7153137 and 5722485 queries of size 300, 400, and 500 respectively. For each query length, we created querysets by randomly choosing 50000 queries. Allowing an edit distance of 5, FANGS was successful in mapping more than 99.98% of the queries in all querysets.

Table 3: Mapping results of FANGS, Bowtie(bowtie-0.9.9.3), Soap(soap2.18) for 50000 queries of length 300, 400 and 500 against the human genome hg19 allowing a maximum edit distance of 2. Third column specifies the sum of the number of mappings found for each query.

Query length	Program	Number of mappings found	Time taken (sec)	Peak virtual memory footprint (gigabytes)	Percentage of reads mapped	Reads mapped per hour
300	FANGS	38962	155.06	4.5	73.81	1160840
	Bowtie	30580	56.00	2.4	41.81	3214285
	Soap	21945	32.67	5.4	43.89	5509641
400	FANGS	30220	175.85	4.5	58.07	1023599
	Bowtie	13172	68.00	2.4	24.67	2647058
	Soap	13378	45.9	5.4	26.76	3921568
500	FANGS	18416	178.59	4.5	35.74	1007895
	Bowtie	5271	70.00	2.4	10.02	2571428
	Soap	5733	48.01	5.4	11.47	3749218

6. CONCLUSIONS

Advances in sequencing techniques necessitate the development of high performance, scalable algorithms to extract biologically relevant information from these datasets. In this paper, we describe a new sequence mapping tool FANGS, which outperforms existing tools in terms of performance and sensitivity when the number of mismatches and gaps is small. While some tools do not allow any gaps or allow a limited number of mismatches and gaps to be specified, FANGS allows a large number of gaps and mismatches to be specified by the user. Moreover, FANGS successfully finds all possible mappings for each query within the specified edit distance with reasonable memory usage. However, as expected, FANGS is not very efficient for larger number of mismatches and insertion/deletions. Moreover, FANGS does not consider quality information while mapping the queries. Both these issues need to be addressed in future work.

With the advent of new technologies, we will need even faster sequence mapping tools to stay at par with the increasing sequencing speed. In addition to developing faster algorithms; we need to use parallel processing, graphics processing unit (GPU) based systems and application specific hardware to build faster sequence mapping tools. The Next Generation Sequencers along with high-speed sequence alignment tools will enable us to fulfill the dream of personal genomics. Just imagine how much this can help in using a patient's DNA in diagnosing a disease or better still, knowing in advance whether a person's DNA encodes a risk of a certain disease.

7. ACKNOWLEDGMENTS

This work was supported in part by NSF CCF-0621443, NSF SDCI OCI-0724599, NSF CNS-0551639 and IIS-0536994, DOE SCIDAC-2: Scientific Data Management Center for Enabling Technologies (CET) grant DE-FC02-07ER25808, DOE SCiDAC award number DE-FC02-01ER25485.

8. REFERENCES

[1] SHRiMP - SHort Read Mapping Package http://compbio.cs.toronto.edu/shrimp/.

[2] Mosaik: http://bioinformatics.bc.edu/marthlab/Mosaik.

[3] BWA: http://maq.sourceforge.net/bwa-man.shtml.

[4] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *J Mol Biol*, 215(3):403–410, October 1990.

[5] H. Jiang and W. H. Wong. Seqmap : mapping massive amount of oligonucleotides to the genome. *Bioinformatics*, pages btn429+, August 2008.

[6] W. J. Kent. Blat–the blast-like alignment tool. *Genome Res*, 12(4):656–664, April 2002.

[7] B. Langmead, C. Trapnell, M. Pop, and S. Salzberg. Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome Biology*, 10(3), 2009.

[8] H. Li, J. Ruan, and R. Durbin. Mapping short dna sequencing reads and calling variants using mapping quality scores. *Genome research*, August 2008.

[9] R. Li, Y. Li, K. Kristiansen, and J. Wang. Soap: short oligonucleotide alignment program. *Bioinformatics*, January 2008.

[10] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, March 1970.

[11] Z. Ning, A. J. Cox, and J. C. Mullikin. Ssaha: A fast search method for large dna databases. *Genome Res.*, 11(10):1725–1729, 2001.

[12] K. L. Patrick. 454 life sciences: Illuminating the future of genome sequencing and personalized medicine. *Yale J Biol Med.*, 80(4):191–4, Dec 2007.

[13] W. R. Pearson and D. J. Lipman. Improved tools for biological sequence comparison. *Proc Natl Acad Sci U S A*, 85(8):2444–2448, April 1988.

[14] P. Pevzner and M. Waterman. Multiple filtration and approximate pattern matching. *Algorithmica*, 13:135–154, 1995.

[15] K. R. Rasmussen, J. Stoye, and E. W. Myers. Efficient q-gram filters for finding all epsilon-matches over a given length. *Journal of Computational Biology*, 13(2):296–308, 2006.

[16] C. Shaffer. Next-generation sequencing outpaces expectations. *Nature Biotechnology*, 25(2):149, February 2007.

[17] A. D. Smith, Z. Xuan, and M. Q. Zhang. Using quality scores and longer reads improves accuracy of solexa read mapping. *BMC Bioinformatics*, 9:128+,

February 2008.

[18] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[19] D. A. Wheeler, M. Srinivasan, M. Egholm, Y. Shen, L. Chen, A. Mcguire, W. He, Y.-J. Chen, V. Makhijani, G. T. Roth, X. Gomes, K. Tartaro, F. Niazi, C. L. Turcotte, G. P. Irzyk, J. R. Lupski, C. Chinault, X.-Z. Song, Y. Liu, Y. Yuan, L. Nazareth, X. Qin, D. M. Muzny, M. Margulies, G. M. Weinstock, R. A. Gibbs, and J. M. Rothberg. The complete genome of an individual by massively parallel dna sequencing. *Nature*, 452(7189):872–876, April 2008.

[20] T. D. Wu and C. K. Watanabe. Gmap: a genomic mapping and alignment program for mrna and est sequences. *Bioinformatics*, 21(9):1859–1875, May 2005.