BIOINFORMATICS ORIGINAL PAPER

Vol. 27 no. 2 2011, pages 189–195
doi:10.1093/bioinformatics/btq648

Sequence analysis

Advance Access publication November 18, 2010

Anatomy of a hash-based long read sequence mapping algorithm for next generation DNA sequencing

Sanchit Misra*, Ankit Agrawal, Wei-keng Liao and Alok Choudhary

Department of Electrical Engineering and Computer Science, Northwestern University, 2145 Sheridan Road, Evanston, IL 60208, USA

Associate Editor: Joaquin Dopazo

ABSTRACT

Motivation: Recently, a number of programs have been proposed for mapping short reads to a reference genome. Many of them are heavily optimized for short-read mapping and hence are very efficient for shorter queries, but that makes them inefficient or not applicable for reads longer than 200 bp. However, many sequencers are already generating longer reads and more are expected to follow. For long read sequence mapping, there are limited options; BLAT, SSAHA2, FANGS and BWA-SW are among the popular ones. However, resequencing and personalized medicine need much faster software to map these long sequencing reads to a reference genome to identify SNPs or rare transcripts.

Results: We present AGILE (AliGnIng Long rEads), a hash table based high-throughput sequence mapping algorithm for longer 454 reads that uses diagonal multiple seed-match criteria, customized q-gram filtering and a dynamic incremental search approach among other heuristics to optimize every step of the mapping process. In our experiments, we observe that AGILE is more accurate than BLAT, and comparable to BWA-SW and SSAHA2. For practical error rates ($<5\%$) and read lengths (200 – 1000 bp), AGILE is significantly faster than BLAT, SSAHA2 and BWA-SW. Even for the other cases, AGILE is comparable to BWA-SW and several times faster than BLAT and SSAHA2.

Availability: http://www.ece.northwestern.edu/~smi539/agile.html.

Contact: smi539@eecs.northwestern.edu

Supplementary information: Supplementary data are available at *Bioinformatics* online.

Received on July 29, 2010; revised on October 27, 2010; accepted on November 14, 2010

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. By mapping these small fragments (reads) to a reference genome, we can sequence the DNA of a new individual. The NGSs are making it possible for these studies to be conducted at a mass scale. These advances will usher an era of personal genomics when each individual can have his/her DNA sequenced and studied to develop more personalized ways of anticipating, diagnosing and treating diseases (Patrick, 2007).

Studies of this nature have already begun. Scientists have found the genetic causes of diseases like Charcot–Marie–Tooth (Lupski *et al.*, 2010) and Miller syndrome (Roach *et al.*, 2010) by sequencing the genomes of patients. These studies have been made possible by plunging costs and increasing speeds of high-throughput sequencing. Next generation sequencers (NGSs) sequence the DNA by generating small substrings of the DNA called reads. With rapid improvements in sequencing technologies, the lengths of the reads are constantly increasing. For example, the lengths of 454 reads increased from about 250 bp in 2007 to about 500 bp in 2009. The length of illumina reads increased from about 30–50 bp in 2007 to about 100 bp in 2009. Pacific Biosciences also announced 1000 bp long reads in 2009. The rate of throughput as well as read lengths of these NGSs are increasing at a pace that puts even the Moore's law to shame. Hence, there is a growing need for tools that can work for longer reads and can still match the pace of the NGSs.

A number of tools have been developed for shorter illumina queries. These include MAQ (Li *et al.*, 2008a), ELAND, SOAP (Li *et al.*, 2008b), BowTie (Langmead *et al.*, 2009), Mosaik, PASS (Campagna *et al.*, 2009), and SHRiMP (Rumble *et al.*, 2009). However, most of these tools work only for read lengths <200. Also, they allow very few number of mismatches (usually <2) and many of them do not allow any gaps. However, as the lengths of reads are rapidly increasing, we need tools that can work for longer read lengths. Moreover, these new tools for longer reads should be able to handle a larger number of gaps and mismatches. To the best of our knowledge, the only other tools specifically designed to work for longer reads are BWA-SW (Li and Durbin, 2010) and FANGS (Misra *et al.*, 2009). BWA-SW is a package based on Burrows–Wheeler Transform (BWT). It supports gapped global alignment with respect to queries and is one of the fastest long read alignment algorithms while also finding suboptimal matches.

Hash tables have been used extensively for short-read mapping and many other related problems (Altschul *et al.*, 1990; Kent, 2002; Li *et al.*, 2008b; Ning *et al.*, 2001; Pearson and Lipman, 1988; Smith *et al.*, 2008). Hence, it may appear that we have exhausted all possible uses of hash tables for sequence mapping. However, we will find that with proper heuristics, hash tables can give excellent speedups for sequence mapping of longer reads as well. The general structure of any hash table-based sequence mapping algorithm is as follows: (i) create a hash table index of the genome; (ii) use the index to find regions in the genome that can potentially be homologous; (iii) examine each region in more detail and output regions that are

*To whom correspondence should be addressed.

indeed homologous. The execution time of existing hash table-based algorithms is dominated by stage (iii). The processing time of this stage is directly proportional to the number of regions found. FANGS is also a hash table-based tool. It uses q-gram filtering and pigeon hole principle to filter out many of the regions to rapidly map 454 reads with nearly 100% sensitivity. However, FANGS is inefficient for error rates greater than 1%. Using techniques in addition to the ones used by FANGS to quickly filter out regions that are not homologous will greatly reduce the time taken. We follow this path.

In this article, we will discuss five different techniques to filter out non-homologous regions and their adaptations to the high-throughput long read sequence mapping problem. Subsequently, we present AGILE—yet another hash table-based tool for sequence mapping—in which we have used these filtering techniques to optimize every step of the sequence mapping process.

2 HIGH-THROUGHPUT SEQUENCE MAPPING

In the context of NGS, 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 refer to the length of S, $S[i]$ to denote the i-th character of S and $S[i:j]$ to denote the substring of S which starts at position i and ends at position j. A *q-gram* of S is defined as a substring of S of length $q>0$. A *q-hit* between two strings S_1 and S_2 is defined as the tuple (x,y) such that $S_1[x:x+q-1]=S_2[y:y+q-1]$. The *unit cost edit distance* (Levenshtein distance) (Rasmussen *et al.*, 2006) between two strings S_1 and S_2 is defined as the minimum number of substitutions, insertions and deletions required to convert S_1 to S_2. We will use $E(S_1,S_2)$ to refer to the *unit cost edit distance* between S_1 and S_2. It can be calculated by using dynamic programming in $O(|S_1||S_2|)$ time (Needleman and Wunsch, 1970; Smith and Waterman, 1981). For a string S, we will refer to the natural decimal representation of S over Σ as $D(S,\Sigma)$. For example, for $\Sigma=\{A,C,G,T\}$, the nucleotides A,C,G,T can be mapped to the numbers $0,1,2,3$, respectively. Therefore:

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

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

This brings us to the formal definition of the sequence mapping problem. We can represent every genomic sequence as a string over the alphabet $\Sigma=\{A,C,G,T\}$. Given a genomic database G of subject sequences $\{S_1,S_2,\cdots,S_l\}$, a query sequence (read) Q of length $|Q|$, the genome sequencing problem is to find the substring α of G that has the minimum value of $E(\alpha,Q)$ for all α. This problem can be reduced to finding the best match α for a query such that $E(\alpha,Q)$ is less than a certain bound. We can keep on increasing the bound till we find a match. Moreover, for a given sequencing error rate, larger reads tend to have more differences in the alignment as compared with shorter reads. Hence, having an absolute bound on the number of differences in an alignment is inappropriate as the length of the reads can vary. The bound should be a fraction of the read length. Hence, we define the ϵ-match sequence mapping problem:

Given a genomic database G of subject sequences $\{S_1,S_2,\ldots,S_l\}$, a query sequence (read) Q of length $|Q|$ and an error rate ϵ find the substring α of G, such that $E(\alpha,Q)$ is minimum and $E(\alpha,Q)\leq\epsilon|Q|$.

3 ALGORITHM

Most sequence mapping algorithms divide the problem into two stages: search stage and alignment stage. The search stage finds regions in the genome that can potentially be homologous to the read. The alignment stage verifies these regions to check if they are indeed homologous. The alignment stage is usually more computationally intensive than the search stage and the time taken is directly proportional to the number of regions found in the search stage. Hence, the best strategy to a fast sequence mapping algorithm is to filter out as many candidate regions as possible before the alignment stage. Various programs try to achieve this with the use of well-designed filters for the specific problem ranges. In this section, we describe the AGILE algorithm that can achieve such filtering for a wide range of read lengths and error rates through a number of carefully designed filters. Some of these filters are generic and work for the entire range of read lengths and error rates, while others cater to a specific subset. However, combining these filters can achieve faster speeds over a wide range.

Table 1. The value of threshold T for different values of error rate ϵ and read length ($q=16$)

Error rate ϵ(%)	Read length				
	100	200	500	1000	10 000
1	2	3	4	4	5
2	1	1	1	2	2
3	0	0	1	1	1
5	0	0	0	0	0
10	0	0	0	0	0

We start by dividing the genome into non-overlapping *q-grams* and storing the locations of the *q-grams* in a *q-gram index* (hash table). Using the *q-gram index*, we find the list of *q-hits* between the read and the genome. Each *q-hit* can be extended on either side to create a candidate region. Regions with a large overlap with each other represent the same alignment and hence can be merged together. This gives us a very large number of regions. In the following subsections, we describe the filtering techniques used in AGILE that we apply to these regions.

3.1 Contiguous perfect matches

Two strings of length m with n differences share a common (exactly matching) substring of length at least $\frac{m}{n+1}$ (Pevzner and Waterman, 1995). FANGS adapted the above formula to note that—given a read Q and genome G, if a substring α of G is such that $E(Q,\alpha)\leq\epsilon|Q|$, then Q and α have a perfectly matching substring of length T *q-grams*, where T is given by:

$$T=\left\lfloor\frac{\lfloor\frac{|Q|}{\epsilon|Q|+1}\rfloor-(q-1)}{q}\right\rfloor$$

where q is the length of the *q-grams*. Therefore, we only consider regions in the genome that have a common substring of length T *q-grams* with the read. This filtering criteria significantly reduces the search space for finding homologous regions. However, as Table 1 demonstrates, the value of T quickly becomes zero beyond a certain percentage of errors. Hence this filtering scheme does not help much if we consider a larger error rate.

3.2 Multiple perfect matches

Kent (2002) had discussed the possibility of using multiple perfect matches as a filter. Each q-hit is a perfect match. The matches need not be contiguous. Let $q=16$. Consider, for example, two q-hits—one starts at position 20 in the read and position 10020 in the genome and second starting at position 52 in the read and position 10052 in the genome. Since there is equal gap between the *q-hits* in the read as well as the genome, they can easily be part of one alignment. Notice that 'genome-position'−'read-position'=10000 for both the reads. This value is called the diagonal. We can filter out a large number of regions by keeping a minimum cutoff on the number of q-hits with equal or slightly different diagonal.

Table 2. The highest value of minimum number of perfect matches to get a particular sensitivity for different values of read lengths and sequence similarity

Read length →	10 000				1000				500				200			
Similarity (%) ↓ Sensitivity →	1	0.9999	0.999	0.99	1	0.9999	0.999	0.99	1	0.9999	0.999	0.99	1	0.9999	0.999	0.99
90	71	81	87	94	0	2	3	5	0	0	0	1	0	0	0	0
95	215	229	237	246	9	13	16	18	2	4	5	7	0	0	1	2
97	324	338	346	355	19	24	26	29	6	9	11	13	0	1	2	3

The first column lists the sequence similarity M. In each subsequent column, we report the maximum value of N for the given read length such that $P^N >$ sensitivity.

Let M be the sequence similarity between the read and the corresponding homologous region in the genome. Assuming that each letter is independent of the previous letter, the probability that a specific *q-gram* in the read matches a *q-gram* in a homologous region in the genome is given by:

$$p_1 = M^q$$

The match of the read in the genome will be of the same length as the read. Number of non-overlapping *q-grams* in the homologous region is given by:

$$K = \lfloor |Q|/q \rfloor$$

The probability that there are exactly n matches in the homologous region is:

$$P_n = \binom{K}{n} {p_1}^n (1-p_1)^{K-n}$$

and the probability that there are N or more matches is the sum:

$$P^N = P_N + P_{N+1} + ... + P_K$$

James Kent discussed the idea of using two perfect matches as a filtering criteria as opposed to here we use a larger number of perfect matches for the same. Table 2 displays the values of N that can be used for different values of sequence similarity and read lengths.

Table 3. Experiments to find the most appropriate values of F and N so as to maximize the number of reads correctly mapped and minimize the number of regions sent to the alignment stage (hence minimize the time taken)

F	N	Correctly mapped	Number of regions	Time taken (CPU in s)
4	1	9978	3567070	1790.73
16	2	9979	140453	166.39
17	2	9980	150242	176.28
18	2	9980	159895	185.91
19	2	9980	170223	196.15
20	2	9981	180430	206.27
28	2	9981	266621	294.38
29	2	9982	279457	307.4
32	3	9969	102282	171.42
64	3	9974	239962	391.9

Of total, 10000 reads of length 1000 were synthetically generated by sampling the human genome. We introduced errors in the reads using an error rate of 10%. For a read length of 1000 and error rate of 10%, $N = 2$ and $F = 17$ work the best. The genome used is human genome.

3.3 Ignore q-grams with high frequency

In both the optimizations above, we are applying theoretical constraints to the *q-hits* between reads and genome in order to filter out unwanted regions. However, some *q-grams* appear much more frequently than many others. As a result, we get a very large number of *q-hits* for some reads. Moreover, if a particular *q-gram* appears very frequently in the genome, then it will produce a lot of matches at undesired places wasting time in processing them. A less frequent *q-gram* is much more useful in pinpointing a match. Hence, our heuristic ignores all *q-grams* with frequency more than a certain cutoff frequency F to save time. To ensure that the contiguous perfect match criteria still works, we give a wildcard to all *q-grams* with frequency greater than F as done in FANGS. In principle, we need to reduce the values of N so that we are still able to find all the homologous regions. A theoretical model may not be accurate unless we take into account the exact probability of each *q-gram*, which makes the model very complex. Hence, we decided to do empirical analysis using synthetic queries for which the correct answers are already known. Table 3 shows results of such experiments for read length 1000. Clearly, for a fixed value of N, higher value of F should result in more regions to process in the alignment stage and more correctly mapped regions. For example, for $N = 1$, even $F = 4$ passes 3567070 regions to the alignment stage. While with $N = 2$ and $F = 17$, AGILE correctly maps more reads and filters out more regions. For $N = 1$, if F is reduced, that will further reduce correctly mapped regions. On the other hand, if F is increased, that will increase the time taken. Comparing $N = 2$ case with $N = 3$ case, even with $F = 64$, $N = 3$ case correctly maps less number of reads but costs more time. Hence for a read length of 1000 and error rate of 10%, $N = 2$ works the best. With $N = 2$ and $F = 17$, AGILE correctly maps 99.8% of the queries. As compared to $F = 17$, $F = 29$ takes 130 extra seconds to increase the number of correctly mapped reads by just 2. Hence, there is a trade-off between the time taken and the number of correctly mapped queries. In this particular case, $N = 2$ and $F = 17$ seems a relatively better choice. Using similar analysis, for a read length of 10000 and error rate of 10%, $N = 32$ and $F = 4$ turns out to be a good choice. This filtering criteria works very well for large read lengths.

3.4 Customized q-gram filtering

Another filter that we have applied in AGILE is keeping a minimum cutoff on the number of q-hits between the read and the region. This is called q-gram filtering. To estimate the effect of q-gram filtering, we conducted experiments with synthetic reads. For each read, we calculated the number of *q-hits* in each region and also whether the region is actually homologous or not. As shown in Figures 1 and 2, homologous regions tend to have larger number of *q-hits* while non-homologous regions tend to have smaller number of q-hits. Hence, with a carefully chosen cutoff on the number of *q-hits*, we can filter out non-homologous regions. However, some queries can have more frequently appearing *q-grams* than others. In that case, those queries with more frequently appearing *q-grams* can have many hits in each region. On the other hand, queries with less frequent *q-grams* can have only a few hits even in a homologous region. Hence, a generic cutoff is not appropriate. In AGILE, we dynamically choose the cutoff for each read. For each read, we find the maximum of the number of *q-hits* in all regions, say C. We keep the cutoff as a fraction f of C. Hence, if a region has $\geq fC$ q-hits, only then it is processed further. The best value of f can filter out the maximum number of non-homologous regions while keeping the number of correctly mapped reads the same. As an example, Table 4 demonstrates the effect of

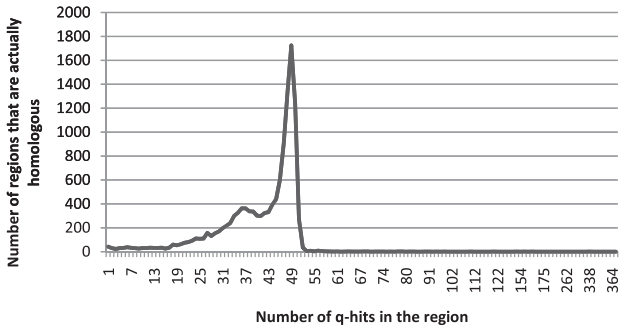

Fig. 1. Histogram of the number of regions that are homologous and number of q-hits they share with the reads. This demonstrates that the homologous regions have larger number of q-hits.

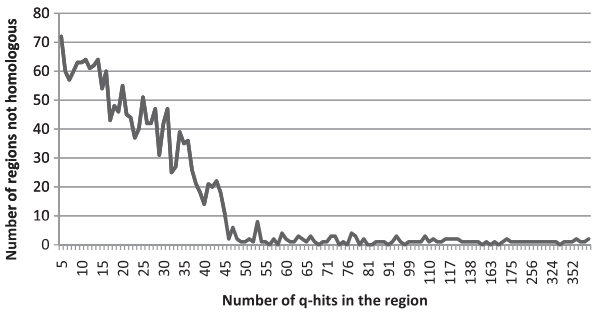

Fig. 2. Histogram of the number of regions that are not homologous and number of q-hits they share with the reads. This demonstrates that the non-homologous regions have smaller number of q-hits.

using various values of f for read length 1000. In this case, the best value of f is 0.5.

3.5 Selecting the error rate ϵ

All the above optimizations assume that we already know ϵ, to filter out the regions. However, the percentage error of a particular read cannot be known in advance. Reads from different sources can have varying error percentages. Even reads from the same source can have variations in error. To account for this, we apply the following heuristics.

For each query, we start with a small value of ϵ, say 3%. We keep increasing the value until we find a match. There are various ways in which the value can be increased. We can increase by 1% each time (e.g. 3, 4, 5, etc.), or by a larger constant interval (e.g. 3, 8, 13, 18, etc.), we can increase the value of the increment by one each time (e.g. 3, 4, 6, 9, 13, etc.), we can double the value of increment each time (e.g. 3, 4, 6, 10, 18, etc.). Through our experiments, we found that doubling the value of increment each time (exponential increment) works faster than the other strategies mentioned above.

Choosing an appropriate starting value of ϵ is extremely crucial for the above step and depends on the error distribution of the reads. If the errors in the reads are distributed in a very small band on the lower side of the spectrum, choosing a higher initial value of ϵ will lead to a lot of extra time wasted in unnecessary processing. On the other hand, if the errors in the reads are distributed over a very small band on the higher side of the spectrum, choosing a smaller initial value of ϵ will mean that we will not find any match for the initial few values of ϵ. Hence, we will waste time in unnecessary and unfruitful processing. In order to 'guess' a good initial value of n to start with, we dynamically adjust the ϵ by learning from the previous queries. We take the average of the edit distance per unit length of all the previous queries as the initial value of ϵ for the next query.

Table 4. Effect of the value of f

f	# Regions	# Regions processed	# Correctly mapped	Time taken (s)
0	180430	180430	9981	206.27
0.1	180430	160168	9981	193.26
0.2	180430	110497	9981	152.44
0.3	180430	90342	9981	133.81
0.4	180430	79593	9981	122.13
0.5	180430	69008	9981	111.45
0.6	180430	52900	9979	96.16
0.7	180430	32728	9975	77.46
0.8	180430	29490	9967	73.92
0.9	180430	28089	9957	72.21
1	180430	27647	9938	71.46

Regions is the number of regions passed as a result of the previous filters. # Regions processed is the number of regions with number of *q-hits* $\geq fC$ for that read. Hence, these are the number of regions processed by the alignment stage. The genome used is human genome. 10000 reads of length 1000 were synthetically generated by sampling the human genome. We introduced errors in the reads using an error rate of 10%. Clearly, $f = 0.5$ works best for this case.

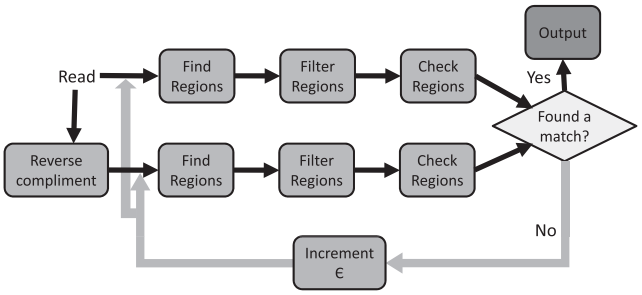

Fig. 3. AGILE workflow.

4 AGILE IMPLEMENTATION

Our implementation takes as input the genome FASTA file and a query FASTA file. The query file typically contains a batch of many queries. In the output, we report locations in the genome where the full length of the read matches along with mapping quality and score of the match.

The high level workflow of AGILE is depicted in Figure 3. AGILE uses the fact that 454 sequencers have a very small error rate (Rothberg and Leamon, 2008). Hence, we divide the problem of finding the best match α of a read $|Q|$ that minimizes $E(\alpha, Q)$ into multiple problems, allowing different values of edit distance. Each such problem is now of the form—find a substring α of G such that $E(\alpha, Q) < \epsilon|Q|$. We start with allowing a small value of ϵ. If we find a match, we output that match and move to the next read. If we do not find a match, we increase the value of ϵ and try again.

AGILE uses a *q-gram index* of the genome to map queries. We process the input reads one by one. Since the read can be from any strand of the DNA, AGILE processes both the read and reverse complement of the read to search for matches. For any sequence, we start by identifying the regions in the reference genome that can potentially be homologous to the read. In the next step, we filter out many of the regions using the heuristics discussed in the previous section. Each homologous region in the filtered list is further processed by using dynamic programming by creating the edit distance matrix to check if the region actually has an edit distance $\leq \epsilon|Q|$.

Table 5. Effect of pruning

Query length	% similarity	Without pruning		With pruning	
		Time taken	Correctly mapped	Time taken	Correctly mapped
1000	98	137.01	9975	77.89	9975
1000	95	160.64	9982	92.84	9982
1000	90	164.52	9981	111.45	9981
10000	98	175.42	1000	144.58	1000
10000	95	205.77	999	188.1	999
10000	90	285.47	1000	270.57	1000

Since the edit distance is bounded by $\epsilon|Q|$, we only need to calculate a diagonal band of width $2\epsilon|Q|+1$ of the matrix. If at any stage, all paths in the edit distance matrix have an edit distance of more than $\epsilon|Q|$, we stop further processing and discard the region. This pruning policy greatly reduces the time taken by the algorithm as many regions are discarded after the first few rows are processed. Table 5 demonstrates the speed benefit obtained by this optimization.

We have run extensive experiments in order to find the most efficient parameter values for different read lengths and ϵ values. Based on these experiments, for each query we automatically set the optimal values of these parameters. Also, as explained in Section 3.5, we dynamically select the most appropriate starting values of ϵ and increment ϵ until we find a match. Automatic selection of parameter values makes it easier for users to use the program as they do not need to worry about deciding the appropriate parameter settings. In addition, having tailored values of parameter settings for different scenarios makes it possible to run real queries that can be of varied lengths and error rates.

5 RESULTS

5.1 Mapping quality calculation

The concept of mapping quality was coined by Li *et al.* (2008a) to estimate the probability that the read sequence has been mapped at the correct place or not. Li and Durbin (2010) approximated the mapping quality as $250(S_1-S_2)/S_1$, where S_1 is the score of the best alignment and S_2 is the score of the second best alignment. We adopt the same approach in our experiments for calculating the mapping quality.

5.2 Results on synthetic data

To create synthetic queries, we used WGSIM script provided in the SAMTOOLS package. The script was modified to adapt to 454 data. We created reads of a total of 10 million bp of different read lengths. For each read length, we introduced 2, 5 and 10% errors. Of total, 20% of the errors were indels. We aligned these simulated reads to the human genome hg19 using AGILE, BWA-SW, BLAT (option -fastMap), SSAHA2 (option -454), Mosaik and PASS. Since these are synthetic reads, we know their coordinates in the genome. Hence, we compared the aligned coordinates to the known coordinates to calculate the alignment error. SSAHA2, BWA-SW and AGILE report mapping quality. However, in the cases when a tool is unable to pick the best and second best alignments, the corresponding mapping quality will be incorrect. Hence, comparing mapping quality values reported by different tools may be invalid as some tools might find a larger number of second best alignments than others. To solve this problem, we calculate mapping quality for each tool using alignments reported by all the tools. For each tool, let S_1 be the score of the best alignment α found by that tool. We take S_2 as the score of the best alignment (other than α) found by any tool. We compute the mapping quality using S_1 and S_2 in the similar manner as described in Section 5.1. For our evaluation, we ran all the experiments on Intel Xeon quad core E5430 2.66 GHz processor with 2×6 MB cache and 32 GB RAM running a Linux-based operating system. Each tool was run in single-threaded mode.

Table 6 shows the CPU time, percentage of confidently (mapping quality >20) aligned reads and percentage of reads incorrectly aligned for AGILE (version 0.3.0), BWA-SW (version 0.5.7), BLAT (version 34) and SSAHA2 (version 2.5.1) for different values of read length and sequencing error rates. We have reported the comparison of AGILE against Mosaik and PASS in the Supplementary Table SI. We used the default command line options for each program unless necessary otherwise. Carefully selected command line options might yield better results.

5.2.1 AGILE vs BWA-SW It is observed from Table 6 that AGILE is more accurate than BWA-SW especially for reads with shorter lengths and more error rates. However, AGILE is also slower than BWA-SW for the same case—reads with shorter lengths and more error rates. For smaller error rates, AGILE is significantly faster than BWA-SW (upto 5 times faster). AGILE is most efficient for read lengths of about 1000. With the rate at which the 454 read lengths are increasing, 1000 bp reads will soon be the norm.

5.2.2 AGILE vs BLAT From Table 6, it is clear that AGILE is significantly faster (upto 30 times faster) than BLAT while still being much more accurate in most cases; most importantly, the cases with larger error rates. Even in cases where AGILE is less accurate than BLAT, it is not much worse. BLAT's lack of accuracy is also due to the '-fastMap' option. However, with default parameters BLAT is more than an order of magnitude slower than with '-fastMap' option.

5.2.3 AGILE vs SSAHA2 AGILE is several times faster than SSAHA2 on all inputs. Also, AGILE is either comparable or more accurate than SSAHA2 for all inputs apart from the case when the read length is 100 and the error rate is 2%. SSAHA2 does not work well for 10 kb reads as it is not designed for such read lengths and thus could not be tested on them.

5.2.4 Memory comparison As far as memory usage is concerned, for mapping reads against human genome, BLAT and BWA-SW use about 4 GB memory. The peak memory requirement of SSAHA2 is about 5.6 GB. For AGILE, the *q-gram index* requires about 3.5 GB memory and the genome itself requires about 3 GB memory. Memory required by each query is insignificant with respect to them.

5.3 Results on real data

It is difficult to evaluate on real data because of the lack of ground truth. However, if two algorithms output the same alignment for a read, it is most likely correct. If two aligners X and Y output different alignments for a read, if X and Y both report low mapping quality, then the alignment is ambiguous and it does not matter which one is wrong. If X reports high mapping quality for a read and X alignment score is worse than or slightly better than Y, X mapping quality is

Table 6. Results on synthetic reads

Read length (bp)	Program	AGILE			BWA-SW			BLAT			SSAHA2		
	Error rate	CPU (s)	Q20%	errAln%	CPU (s)	Q20%	errAln%	CPU (s)	Q20%	errAln%	CPU (s)	Q20%	errAln%
100	2	112.75	93.39	5.05	161	93.66	4.74	430.54	89.41	16.37	2014.92	93.94	2.05
	5	191.94	91.62	2.94	135	90.63	12.8	327.2	78.75	51.71	2857.75	93.9	4.42
	10	350.93	79.54	8.98	104	77.2	38.03	261.96	62.56	86.47	3567.85	92.22	8.45
200	2	77.27	95.54	0.72	223	95.68	4.43	565.83	95.52	2.57	1064.29	95.43	3.57
	5	142.06	95.36	1.01	189	95.41	8.39	392.38	92.99	19.64	1049.98	95.51	7.02
	10	395.06	92.83	2.42	145	91.39	17.7	278.78	78.95	68.39	1613.21	95.54	10.14
500	2	56.97	97.2	0.22	277	97.2	0.53	900.65	97.19	0.01	1256.23	96.78	0.47
	5	111.39	97.33	0.37	205	97.34	0.69	583.64	97.19	0.83	1078.03	96.67	0.74
	10	277.72	96.73	0.61	160	96.59	1.4	326.71	93.87	32.76	844.04	96.06	0.95
1000	2	69.34	97.38	0.26	243	97.38	0.4	1107.92	97.37	0.01	1585.27	97.04	0.34
	5	78.38	98.02	0.23	201	98.02	0.35	808.55	97.96	0.07	1359.22	97.02	1.02
	10	86.52	97.56	0.19	135	97.52	0.5	392.9	96.09	9.93	1077.6	95.77	1.89
10000	2	122.57	98.2	0.1	160	98.2	0.1	3016.57	98.2	0	–	–	–
	5	139.58	99	0.1	136	99	0.1	1654.69	99	0	–	–	–
	10	197.78	98.2	0.1	125	98.2	0	757.92	98.2	0	–	–	–

We created reads of a total of 10 million bp of different read lengths. Q20%, percentage of reads with mapping quality >20; errAln%, percentage of reads incorrectly aligned.

Table 7. Breakup of alignments that are mapped inconsistently between BWA-SW and AGILE

Condition	Count	Average read	BWA-SW		AGILE	
			Average mapping length quality	Average difference	Average mapping quality	Average difference (%)
BWA-SW≥20; AGILE unmapped	46	285.89	122.61	11.21	–	–
BWA-SW≥20 plausible; AGILE<20	49	250.31	60.53	5.65	3.16	12.1
BWA-SW≥20 questionable	6	268.33	25.5	5.45	125	6.32
AGILE≥20; BWA-SW unmapped	203	140.34	–	–	250	9.46
AGILE≥20 plausible; BWA-SW<20	247	175.3	1.93	11.8	249.07	8.23
AGILE≥20 questionable	413	273.69	0.61	3.95	250	3.45

We mapped 100000 reads uniformly selected from SRR005010 (pre-filtered to remove any reads smaller than 100 bp) against the human genome hg19 with BWA-SW and AGILE, respectively. We call a read inconsistently mapped if the left most position of the alignments found by BWA-SW and AGILE differ by more than the length of the read. For each alignment, we calculated the score [number of matches—three times the number of differences (edit distance)]. We call a AGILE alignment plausible if 250 (AGILE score − BWA-SW score)/(AGILE score)≥20 (i.e. using BWA alignment as the next best alignment, AGILE mapping quality >20). Essentially, this means that the AGILE alignment is sufficiently better. Otherwise, we call an AGILE alignment questionable. We use similar definitions of plausible and questionable for BWA-SW alignments. In adddition, 'AGILE≥20' is defined as AGILE alignments with mapping quality ≥20.

wrong and X is not aware of an equally probable alignment. This analysis is similar to the one reported in Li and Durbin (2010).

We ran BWA-SW and AGILE on real queries obtained from the NCBI short read archive SRR005010. We filtered the read set to remove queries smaller than 100 bp long and uniformly selected 100k read with an average length of 338 bp. AGILE took 283 CPU seconds and BWA-SW took 939 CPU seconds. Both tools found 96293 common alignments. Of total, 357 reads were not mapped by any of the tools. Table 7 shows breakup of reads that are aligned by only one aligner or mapped to different places, and are assigned mapping quality ≥20 for either BWA-SW or AGILE. Overall, BWA-SW misses 203+247=450 alignments that AGILE maps well. AGILE misses 95 (46+49) alignments that are aligned well by BWA-SW. Note that, even though the average read length of the entire read set is 338, the inconsistencies come mostly in the case of smaller read lengths (average length 286 or smaller). This is in accordance with the results on synthetic reads as both aligners tend to make more mistakes in aligning reads of shorter lengths. BWA-SW tends to miss more alignments in even shorter (140–170) read length range, while AGILE misses more alignments for a bit longer (250–286) reads.

6 ANALYSIS

AGILE is similar to BLAT, SSAHA2, FANGS, Mosaik, PASS and BWA-SW in sharing the seed and extend strategy to get candidate regions. However, the major difference lies in the way heuristics are employed to reduce the number of regions to be processed. BLAT and SSAHA2 consider short (10–15 bp) exact matches as seeds. BLAT also provides functionality to use short inexact match

(one mismatch allowed) or two exact matches slight differing in diagonal values. For longer reads, both these techniques result in a very large number of candidate regions. AGILE filters the candidates by allowing longer seeds (upto 28 bp). It also uses a cutoff on the minimum number of contiguous exact seed matches and larger cutoff on the minimum number of exact matches with equal or slightly different diagonal values. To the best of our knowledge, no other tool uses multiple perfect match filtering criteria with a cutoff of more than two matches. This is similar to using a long gapped seed. BWA-SW also uses long gapped seeds for the similar reason. The main difference between BWA-SW and AGILE is the way long gapped seed is implemented. While BWA-SW uses Prefix Trie and Prefix DAWG, AGILE relies on a much simpler data structure *q-gram index* (hash table) and diagonal coordinates. Both BWA-SW and AGILE further use different heuristics in order to reduce the search space.

The heuristics used by AGILE are similar to the ones used in string matching or sequence mapping algorithms. For each heuristic, AGILE adapts it to the problem of long read mapping. Customized q-gram filtering is an example. Q-gram filtering is a well-known technique for filtering out unwanted regions. However to the best of our knowledge, no algorithm has used q-gram filtering with the cutoff customized for each read. Another major contribution of AGILE is the gradual increase of the allowed error rate till we find a match. Most tools use the allowed error rate to decide thresholds for pruning the search space. However, many reads have a small number of errors. Since these tools use a fixed value of the allowed error rate, they end up using a larger value of the allowed error rate for all reads in order to also map the reads with larger errors, resulting in a loss of efficiency. Hence, gradually increasing the allowed error rate in mapping can have tremendous effects on the efficiency of these tools.

7 CONCLUSION

Advances in sequencing techniques necessitate the development of high performance, scalable algorithms to extract biologically relevant information from these datasets. Research on developing sequence mapping algorithms has been largely focused on mapping short reads, and little work has been done for longer 454 reads. AGILE is a hash-based sequence mapping algorithm that rapidly maps long reads using efficient heuristics to optimize different steps of the mapping process. AGILE can handle very large genome sizes and read lengths. It is flexible in that it allows a large number of mismatches and insertions/deletions in mapping and provides command line parameters to control every step of the mapping process. The best sensitivity and specificity of AGILE is achieved when the reads are longer and the error rate is small. Considering that with the improvement in sequencing technology, the read lengths will increase further and the error rates will decrease, AGILE should be even more useful in the future.

Funding: This work was supported in part by National Science Foundation award numbers: CCF-0621443, SDCI OCI-0724599, CNS-0551639, IIS-0536994, and HECURA-0938000. This work was also partially supported by Department of Energy grants DE-FC02-07ER25808 and DE-FG02-08ER25848.

Conflict of Interest: none declared.

REFERENCES

Altschul,S.F. *et al.* (1990) Basic local alignment search tool. *J. Mol. Biol.*, **215**, 403–410.
Campagna,D. *et al.* (2009) Pass: a program to align short sequences. *Bioinformatics*, **25**, 967–968.
Kent,W.J. (2002) Blat–the blast-like alignment tool. *Genome Res.*, **12**, 656–664.
Langmead,B. *et al.* (2009) Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome Biol.*, **10**, R25+.
Li,H. and Durbin,R. (2010) Fast and accurate long read alignment with burrows-wheeler transform. *Bioinformatics*, **26**, 589–595.
Li,H. *et al.* (2008a) Mapping short dna sequencing reads and calling variants using mapping quality scores. *Genome Res.*, **18**, 1851–1858.
Li,R. *et al.* (2008b) Soap: short oligonucleotide alignment program. *Bioinformatics*, **24**, 713–714.
Lupski,J.R. *et al.* (2010) Whole-genome sequencing in a patient with charcot-marie-tooth neuropathy. *N. Engl. J. Med.*, **362**, 1181–1191.
Misra,S. *et al.* (2009) Fangs: high speed sequence mapping for next generation sequencers. In *Proceedings of ACM Symposium of Applied Computing (ACM SAC)*, Sierre, Switzerland.
Needleman,S.B. and Wunsch,C.D. (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.*, **48**, 443–453.
Ning,Z. *et al.* (2001) Ssaha: a fast search method for large dna databases. *Genome Res.*, **11**, 1725–1729.
Patrick,K.L. (2007) 454 life sciences: illuminating the future of genome sequencing and personalized medicine. *Yale J. Biol. Med.*, **80**, 191–194.
Pearson,W.R. and Lipman,D.J. (1988) Improved tools for biological sequence comparison. *Proc. Natl Acad. Sci. USA*, **85**, 2444–2448.
Pevzner,P. and Waterman,M. (1995) Multiple filtration and approximate pattern matching. *Algorithmica*, **13**, 135–154.
Rasmussen,K.R. *et al.* (2006) Efficient q-gram filters for finding all epsilon-matches over a given length. *J. Comput. Biol.*, **13**, 296–308.
Roach,J.C. *et al.* (2010) Analysis of genetic inheritance in a family quartet by whole-genome sequencing. *Science*, **328**, 636–639.
Rothberg,J.M. and Leamon,J.H. (2008) The development and impact of 454 sequencing. *Nat. Biotechnol.*, **26**, 1117–1124.
Rumble,S.M. *et al.* (2009) Shrimp: accurate mapping of short color-space reads. *PLoS Comput. Biol.*, **5**, e1000386.
Smith,T.F. and Waterman,M.S. (1981) Identification of common molecular subsequences. *J. Mol. Biol.*, **147**, 195–197.
Smith,A.D. *et al.* (2008) Using quality scores and longer reads improves accuracy of solexa read mapping. *BMC Bioinformatics*, **9**, 128.