Homework 1

Yunyang Li
Bioinformatics

April 8, 2020

1 Bedtools

Ideally, we can firstly generate 5 random sequences from bedtools by the following commands:

```
>> bedtools random -l 10000 -n 5 -g hg38.genome > random38.bed
>> bedtools getfasta -fi ./hg38.fa  -bed random38.bed -name -fo
   my_seq.fa
```

Remark: The first step was intended to generate random sequences from the hg38 database. It's kind of like the first step when we are constructing gene library, we break down the whole genome into fragments and we sequence these fragments separately. Normally, the fragments ranges from 2-10kb. In here, we set the fragments length to be 10000. Also, this procedure serves as a significant purpose in the later methodology. It adds a fingerprint to itself, making us able to recognize which fragment it came from thereafter.

The format generated by bedtools was primarily in 'bed'. The second step was to convert the bed file into a fasta file using a built-in function in bedtools. **Notice that the getfasta function gives every sequence a particular name, which specifies the region of that simulated sequence in bed style. In this way, we have the "right" answer and can use it to determine if a read is placed correctly.**

2 Wgsim

The next step is to generate simulated reads from the DNA fragments. Wgsim is a perfect match for that purpose.[1].

2.1 Good data

These reads should be properly aligned in the final evaluation.(i.e. equals the sum of true positive and false negative)

```
wgsim my_seq.fa out.read1.fq out.read2.fq
```

Remark: The parameters for simulation are listed as follows[1]:

- base error rate [0.020]
- outer distance between two reads [500]
- number of read pairs [1000000]
- length of the first read [70]
- length of the second read [70]
- rate of mutations [0.0010]
- fraction of indels [0.15]

The Illumina/Solexa sequencing technology typically produces 50–200 million 32–100bp reads on a single run of the machine.[2] And the DNA fragments are usually 400-600bp. The simulated fragment length are around 500. So the default parameters work fine to simulate the data in practice from sequencing. We can view these parameters as the experimental errors, and a good aligner tool should be able to ignore these minor errors and still give the correct alignment results(i.e. categorize these data as true).

In order to compare the performances between different softwares on different read lengths, run the following commands with different parameters on -1 and -2:

```
wgsim -1 32 -2 32 my_seq.fa len30.1.fq len30.2.fq
```

2.2 Bad data

These data are created in order to test the functionality of the algorithms when the data is not ought to be aligned. In other words, these data should be categorized as the sum of false positive and true negative.

```
wgsim my_seq.fa -e 1.0 error.out1.fq error.out2.fq
```

Remark: The parameters for simulation are listed as follows[1]:

- base error rate [**1.000**]
- outer distance between two reads [500]
- number of read pairs [1000000]
- length of the first read [70]
- length of the second read [70]
- rate of mutations [0.0010]
- fraction of indels [0.15]

Also generating the bad data of different read length

```
wgsim -1 32 -2 32 -e 1.0 my_seq.fa len30er.1.fq len30er.2.fq
```

An alternative way of generating bad data could be implemented with Python, as opposed to directly wielding wgsim:

```python
import random
def random_sequence(seq_len):
    ret_seq = ''
    my_dic = {0:'A',1:'T',2:'G',3:'C'}
    for i in range(seq_len):
        ret_seq += my_dic[random.randint(0,3)]
    return ret_seq

def fa_generator(file_name,n,seq_len): # n is the number of sequence
    with open(file_name, 'w+') as f:
        for i in range(n):
            ret_seq = random_sequence(seq_len)
            starting_index = random.randint(1,1000000000000) # randomly simulate a starting index
            f.write ('>{0}::chr{1}:{2}:{3}\n'.format(i+1, random.randint(1,50), starting_index, starting_index+seq_len))
            f.write(ret_seq+'\n')
```

Remark: This script simulates a sequence generated all by random. Hence, the probability that it is aligned to human genome could be very low (**because it excludes paralogs and orthologs**). After running this code, we could send the generated fasta file back to wgsim, running the same workflow as the 'Good Data'.

3 BWA

The first tool chosen to be compared was **BWA**. Firstly, we construct the index of BWA(generating BW Transform).

```
bwa index -a bwtsw hg38.fa
```

Remark: I've chosen the whole human genome as references, of which the file size is more than 3 gigabytes. According to the BWA specifications[3], the default algorithm for indexing doesn't fit anymore. In light of that, I specify the algorithm option to be **bwtsw** which is a better match for larger databases.

3.1 BWA-MEM

Apply BWA-MEM by following commands:

```
bwa mem hg38.fa out.read1.fq out.read2.fq > ret.sam
bwa mem hg38.fa error.read1.fq error.read2.fq > error.sam
```

Checking out the alignment results:

```
>>samtools flagstat ret.sam
2000003 + 0 in total (QC-passed reads + QC-failed reads)
0 + 0 secondary
3 + 0 supplementary
0 + 0 duplicates
1999964 + 0 mapped (100.00% : N/A)
2000000 + 0 paired in sequencing
1000000 + 0 read1
1000000 + 0 read2
1999762 + 0 properly paired (99.99% : N/A)
1999942 + 0 with itself and mate mapped
19 + 0 singletons (0.00% : N/A)
170 + 0 with mate mapped to a different chr
33 + 0 with mate mapped to a different chr (mapQ>=5)
```

The same procedure was also done on 'error.sam'.

```
>>samtools flagstat error.sam
2000994 + 0 in total (QC-passed reads + QC-failed reads)
0 + 0 secondary
994 + 0 supplementary
0 + 0 duplicates
22276 + 0 mapped (1.11% : N/A)
2000000 + 0 paired in sequencing
1000000 + 0 read1
1000000 + 0 read2
0 + 0 properly paired (0.00% : N/A)
0 + 0 with itself and mate mapped
21282 + 0 singletons (1.06% : N/A)
0 + 0 with mate mapped to a different chr
0 + 0 with mate mapped to a different chr (mapQ>=5)
```

The recall and precision could be calculated as follows:

$$recall = \frac{true\ positive}{true\ positive + false\ negative} = \frac{1999964}{2000003} = 99.998\%$$

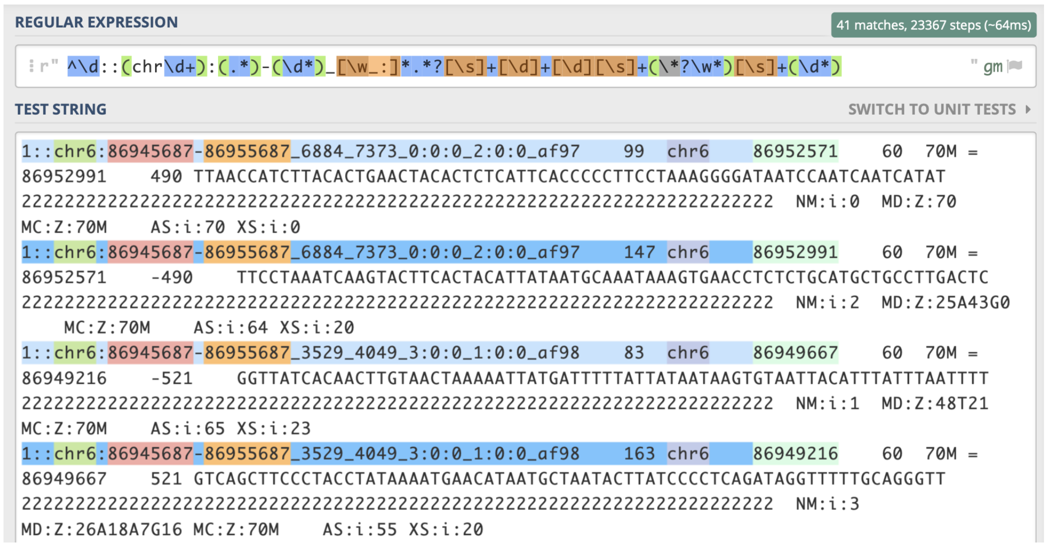
$$precision = \frac{true\ positive}{true\ positive + true\ negative} = \frac{1999964}{1999964 + 22276} = 98.898\%$$

And that is just a rough estimation for this calculation. In general, there might be some cases that those who are mapped by the software are mapped to an incorrect place. Thus, a more general method could be implemented to fix this problem. Notice that previously in bedtools, we have recorded the the core derivation of each simulated reads, including which chromosome was it generated and in what positional range is this simulated read(See Fig1). We can compare this information with the aligned results to give a more stringent verification by the following python scripts to handle sam file.

```python
import re
import pandas as pd
import numpy as np
def readsam(filename):
    col1, col2, col3, col4, col5 = [], [], [], [], []
    with open(filename, 'r') as f:
        while True:
            to_be_processed=f.readline()
            if not to_be_processed: # if the end of the file
                break
            if to_be_processed[0] == '@': # we only focus on lines that not start with @
                continue
            try:
                search_obj = re.search(r"^\d::(chr\d+):(.*)-(\d*)_[\w_:]*.*?[\s]+[\d]+[\d][\s]+(\*?\w*)[\s]+(\d*)",to_be_processed)   # regex
                col1.append(search_obj.group(1))
                col2.append(search_obj.group(2))
                col3.append(search_obj.group(3))
                col4.append(search_obj.group(4))
                col5.append(search_obj.group(5))
            except AttributeError:
                print(to_be_processed)

        col1 = pd.Series(col1)
        col2 = pd.Series(col2)
        col3 = pd.Series(col3)
        col4 = pd.Series(col4)
        col5 = pd.Series(col5)
    return pd.DataFrame({"sim_chr":col1,"sim_start":col2,"sim_end":col3,"ali_chr":col4,"ali_start":col5})
```

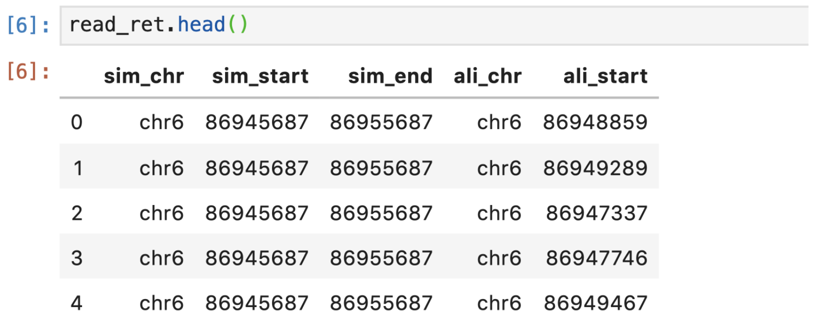
Remark: The regular expression was tested by a regex debugger with over 1000 test cases.[4] Special cases where the the aligned chromosome is '*' are also considered (graded by gray). When running the program over millions of lines, no "to be processed string" was printed(i.e. No AttributeError was raised), which testifies the solidity of this regular expression.



```
r" ^\d::(chr\d+):(.*)-(\d*)_[\w_:]*.*?[\s]+[\d]+[\d][\s]+(\*?\w*)[\s]+(\d*) " gm
```

```
1::chr6:86945687-86955687_6884_7373_0:0:0_2:0:0_af97     99  chr6      86952571     60   70M =
86952991     490 TTAACCATCTTACACTGAACTACACTCTCATTCACCCCCTTCCTAAAGGGGATAATCCAATCAATCATAT
2222222222222222222222222222222222222222222222222222222222222222222222  NM:i:0  MD:Z:70
MC:Z:70M     AS:i:70 XS:i:0
1::chr6:86945687-86955687_6884_7373_0:0:0_2:0:0_af97     147 chr6      86952991     60   70M =
86952571     -490    TTCCTAAATCAAGTACTTCACTACATTATAATGCAAATAAAGTGAACCTCTCTGCATGCTGCCTTGACTC
2222222222222222222222222222222222222222222222222222222222222222222222  NM:i:2  MD:Z:25A43G0
    MC:Z:70M     AS:i:64 XS:i:20
1::chr6:86945687-86955687_3529_4049_3:0:0_1:0:0_af98     83  chr6      86949667     60   70M =
86949216     -521    GGTTATCACAACTTGTAACTAAAAATTATGATTTTTATTATAATAAGTGTAATTACATTTATTTAATTTT
2222222222222222222222222222222222222222222222222222222222222222222222  NM:i:1  MD:Z:48T21
MC:Z:70M     AS:i:65 XS:i:23
1::chr6:86945687-86955687_3529_4049_3:0:0_1:0:0_af98     163 chr6      86949216     60   70M =
86949667     521 GTCAGCTTCCCTACCTATAAAATGAACATAATGCTAATACTTATCCCCTCAGATAGGTTTTTGCAGGGTT
2222222222222222222222222222222222222222222222222222222222222222222222  NM:i:3
MD:Z:26A18A7G16 MC:Z:70M     AS:i:55 XS:i:20
```

Figure 1: the color coding of the regular expression



```
[6]: read_ret.head()
```

[6]:

	sim_chr	sim_start	sim_end	ali_chr	ali_start
0	chr6	86945687	86955687	chr6	86948859
1	chr6	86945687	86955687	chr6	86949289
2	chr6	86945687	86955687	chr6	86947337
3	chr6	86945687	86955687	chr6	86947746
4	chr6	86945687	86955687	chr6	86949467

Figure 2: the captured groups are listed as Numpy DataFrame

Restrictions are listed as follows:

- The chromosome aligned should be the same as the original simulated chromosome.
- The aligned position should be in the range of the simulated starting and ending position.

Finally, execute the following codes:

```
read_ret=readsam("/Users/clintli/my_seq_2.sam")
true_positive = sum((read_ret['sim_start'] <= read_ret['ali_start
    ']) & (read_ret['ali_start'] <= read_ret['sim_end']) & (
    read_ret['sim_chr'] == read_ret['ali_chr'])) # count the reads
     that satisfie the restrictions.
```

4 Bowtie

Firstly, the reference genome should be converted to a bowtie index file. The file could be downloaded from the bowtie manual[5], or be built locally.

Run the following command:

```
bowtie -p 8 -S ~/Downloads/bowtie/GCA_000001405.15
    _GRCh38_no_alt_analysis_set.fna.bowtie_index -1 out.read1.fq
    -2 out.read2.fq bowtie.sam
```

5 Bowtie2

The index file for bowtie(Ver 1.2.3) and bowtie2 could be used interchangeably[5]. So we could directly run the following command:

```
bowtie2 -p 10 -x ~/Downloads/bowtie/GCA_000001405.15
    _GRCh38_no_alt_analysis_set.fna.bowtie_index -1 out.read1.fq
    -2 out.read2.fq | samtools sort -O bam -@ 10 -o - > output.bam
```

Remark: -p is related to threads NTHREADS. -x specify the botwie index's directory *You have to specify the name of the genome index in order to make it work.* After finishing bowtie2, I piped it to samtools to convert it to a bam file.

6 Subread

Following the similar procedure as bwa, first construct the index

```
subread-buildindex -o my_index hg38.fa
```

Then run the aligner on the simulated data

```
subread-align -t 1 -T 5 -i my_index -r ./out.read1.fq  -R ./out.
    read2.fq  -o output_subread.bam
```

Remark:

- -t specify the Datatype (0 represents RNA-seq, 1 represents DNA seq)
- -i is the name of the gene index
- -T number of threads
- -r first reads
- -R second reads

7 Initial data

Remark: Bowtie was not categorized. Because bowtie fails to give me the right alignment results initially. The later on parameter tuning will handle this. The reason why we only compare length 30 and length 70 is that we observe some abnormal situation, and we will compare more read lengths later.

	Correct Align	Incorrect Align	Total Align	Unaligned	Total
BWA-MEM	1999657	307	1999964	39	2000003
Bowtie2	1922612	33183	1955795	44205	2000000
Subread	1792731	36632	1829363	170637	2000000

	Len 30 - precision	Len 30 - Recall	Len-70 precision	Len-70-Recall
BWA MEM	99.330%	93.737%	98.890%	99.998%
Bowtie2	97.021%	94.935%	99.775%	97.790%
Subreads	99.723%	70.327%	99.278%	91.468%

8 Parameter tuning

8.1 Subread

Notice that in our performance evaluation, the subread exhibited abnormal recall. The reason could be wrong choices of parameters or malfunction of the software.

Firstly, the the consensus threshold, specified by -m, is the minimal number of consensus subreads required for reporting a hit. [6]. The default value of m is 3. Initially, we tune the m to be 2.

```
subread-align -t 1 -T 5 -m 2 -i my_index -r ./len30.1.fq  -R ./
   len30.2.fq  -o output_subread.bam
```

The revised results are listed as follows:

```
Good Data Summary
Total fragments : 1000000
Mapped : 916269 (91.6%)
```

Compared with 70.327% previously, this is a big improvement. Under this circumstances, we also need to calculate the precision to see if there is a plummet in precision performances. The bad data results are listed under:

```
Bad Data Summary
Total fragments : 1000000
Mapped : 326068 (32.6%)
```

The revised precision is 73.754%. Clearly, the precision has decreased a lot if we tune the -m. It could be explicated that the -m sets the standard for a hit, if we set it too high, the alignment requirements get very high, the recall will fall. If we set it too low, the bad data could be easily aligned, the precision will fall. Hence, in practice, it is very important to balance the precision and recall. But in this case, tuning m lower didn't make our things better. We need to find another way out.

Secondly, -d specify the minimum fragment/template length, 50 by default. Note that if the two reads from the same pair do not satisfy the fragment length criteria, they will be mapped individually as if they were single-end reads. [6] In wgsim, we specify the read length of data to be 30, which is below this default threshold. We need to revise that.

```
subread-align -t 1 -T 5 -d 30 -i my_index -r ./out.read1.fq  -R
    ./out.read2.fq  -o output_subread.bam
```

This command yields a way better result than previous commands

```
Good Data Summary
Total fragments : 1000000
Mapped : 842078 (84.2%)
```

And the precision doesn't change much

```
Bad Data Summary
Total fragments : 1000000
Mapped : 37869 (3.8%)
```

8.2 bowtie

Previously, when using bowtie, in paired-end mode, the alignment rate was 0 initially. The odds are if we use single-end mode for each separate reads, the alignment rates were both above 90%.

After searching for similar circumstances, chances are bowtie defaults to align two reads that the first read is forward and the second read is reverse, and your input might not be the case. But after adding -ff, -rf, -fr separately (which stands for forward forward, reverse forward, forward reverse), the alignment rate is still 0.

Another advice could be the stringent default requirements of insert size under paired-end mode.[5]

```
-X --maxins <int>
The maximum insert size for valid paired-end alignments. E.g. if
    -X 100 is specified and a paired-end alignment consists of two
     20-bp alignments in the proper orientation with a 60-bp gap
    between them, that alignment is considered valid (as long as -
    I is also satisfied). A 61-bp gap would not be valid in that
    case. If trimming options -3 or -5 are also used, the -X
    constraint is applied with respect to the untrimmed mates, not
     the trimmed mates. Default: 250.
```

In wgsim, the default outer distance between two reads was 500, so 250 was too short for that. For safety reasons, adjust that to 1000

```
bowtie --sam -X 1000 -p 8  GCA_000001405.15
    _GRCh38_no_alt_analysis_set.fna.bowtie_index -1 out.read1.fq
    -2 out.read2.fq  bowtie.sam
```

The alignment results were improved a lot by this adjustment.

```
# reads processed: 1000000
# reads with at least one reported alignment: 842910 (84.29%)
# reads that failed to align: 157090 (15.71%)
Reported 842910 paired-end alignments
```

By the same approach, we need to verify the conservation of precision. Running bowtie on bad data yields:

```
» bowtie --sam -X 1000 -p 8  GCA_000001405.15
    _GRCh38_no_alt_analysis_set.fna.bowtie_index -1 error.out1.fq
    -2 error.out2.fq  bowtie.err.sam
# reads processed: 1000000
# reads with at least one reported alignment: 6 (0.00%)
# reads that failed to align: 999994 (100.00%)
Reported 6 paired-end alignments
```

8.3 Bowtie2

After inspecting the results of bowtie2, we also notice some abnormal results.

```
>>>samtools flagstat ret.sam
[W::sam_read1] Parse error at line 2000198
[bam_flagstat_core] Truncated file? Continue anyway.
2000000 + 0 in total (QC-passed reads + QC-failed reads)
0 + 0 secondary
0 + 0 supplementary
0 + 0 duplicates
1955795 + 0 mapped (97.79% : N/A)
2000000 + 0 paired in sequencing
1000000 + 0 read1
1000000 + 0 read2
998970 + 0 properly paired (49.95% : N/A)
1931340 + 0 with itself and mate mapped
24455 + 0 singletons (1.22% : N/A)
104132 + 0 with mate mapped to a different chr
57674 + 0 with mate mapped to a different chr (mapQ>=5)
```

Although most of the reads were aligned, but most of them were not reported properly paired by bowtie2. The reason that accounts for this is the same as bowtie1. Only if the distance between the two paired-end reads were within the range of setting can it report a properly paired. So fix this by tuning -X(the same as bowtie):

```
bowtie2 -p 10 -X 1000 -x ~/Downloads/bowtie2/GCA_000001405.15
    _GRCh38_no_alt_analysis_set.fna.bowtie_index  -1 out.read1.fq
      -2 out.read2.fq  >> bowtie2.sam
```

Remark: The default setting for bowtie2 is 500. So we could see that, previously in bowtie, if we set the parameter to be 500, it will be unsafe. The reason why this occur could be that 500 is right on the threshold, chances are half of the alignment(50.05% or so) will fall out of range because of some insertions or mutations, which results in the distance between two alignments to increase.

The parse error vanishes, and the number of properly paired seems correct.

```
» samtools flagstat  bowtie2.sam
2000000 + 0 in total (QC-passed reads + QC-failed reads)
0 + 0 secondary
0 + 0 supplementary
0 + 0 duplicates
1978652 + 0 mapped (98.93% : N/A)
2000000 + 0 paired in sequencing
1000000 + 0 read1
1000000 + 0 read2
1977050 + 0 properly paired (98.85% : N/A)
1977054 + 0 with itself and mate mapped
1598 + 0 singletons (0.08% : N/A)
4 + 0 with mate mapped to a different chr
0 + 0 with mate mapped to a different chr (mapQ>=5)
```

As in bowtie2 and bowtie manual specifies, if we set -X to be fairly large, it will seriously affect the speed of the program.[5]In order to conduct performance evaluation, we need to find the minimum value of X that can give the proper alignment results. So we set -X to 500,600,650,700,800 to inspect the lowest value that can reach the maximum paired rate. Also, in order to verify this desirable X doesn't vary with read length, we compare length 70 and length 150.

From the data, we can roughly say that 650 is our desirable results. If you want to obtain a more precise result, you can continue to bisect in the range (600,650). Whereas here, the requirements are not that stringent, so we will choose 650 as our parameter in the later performance evaluation.

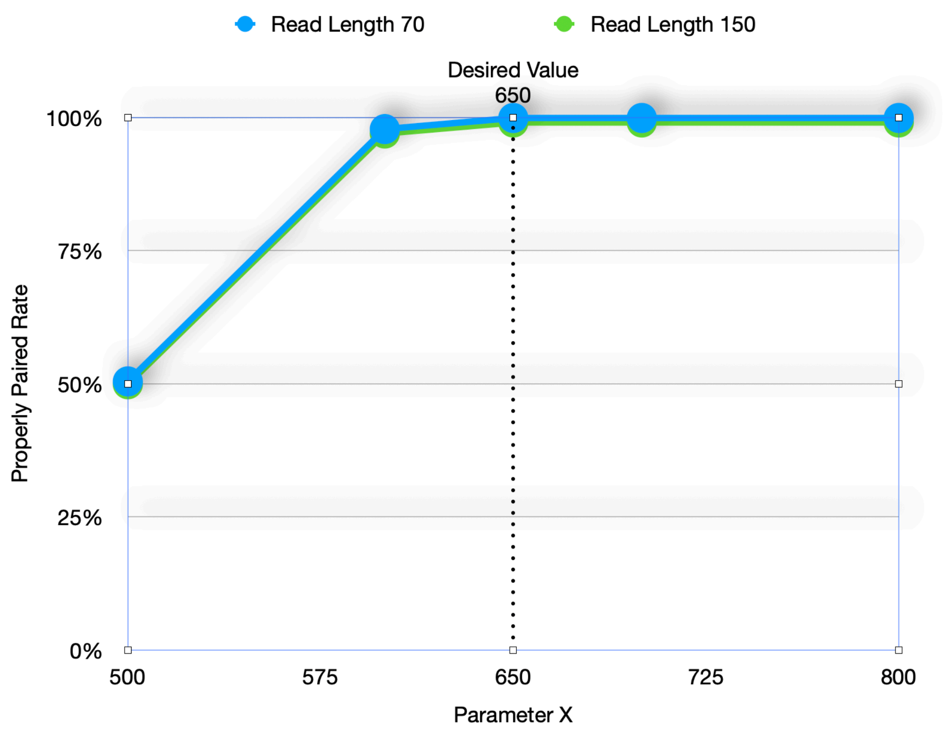


Figure 3: Properly paired rate varies with X

9 Performance Evaluation

9.1 Methodology

To weave every step I've done into an integrated workflow, I wrote a short bash script to conduct the who process. The input varies with read length and number of threads used for concurrency, the output involves time consumption, precision, recall of each software. For further details, see the attached Readme.md which clarifies the usage and functionality of the script.

9.2 Speed

In order to evaluate time, we need to control different parameters (number of threads/length of reads) For each software, we set the read length to be 30,50,70,100,150 and the number of threads is 8.

	Bowtie	Subread	Bwa Mem	Bowtie2
30	28s 2ms	35s 592ms	1m 18s 875ms	1m 10s 497ms
70	35s 244ms	50s 279ms	1m 7s 888ms	1m 2s 377ms
50	27s 758ms	44s 918ms	1m 1s 121ms	55s 713ms
100	50s 8ms	1m 2s 271ms	1m 31s 762ms	1m 18s 916ms
150	1m 5s 721ms	1m 8s 940ms	1m 33s 415ms	2m 10s 774ms
175	1m 16s 417ms	1m 40s 498ms	1m 36s 882ms	2m 11s 433ms
200	1m 18s 866ms	1m 48s 501ms	1m 51s 893ms	2m 41s 503ms

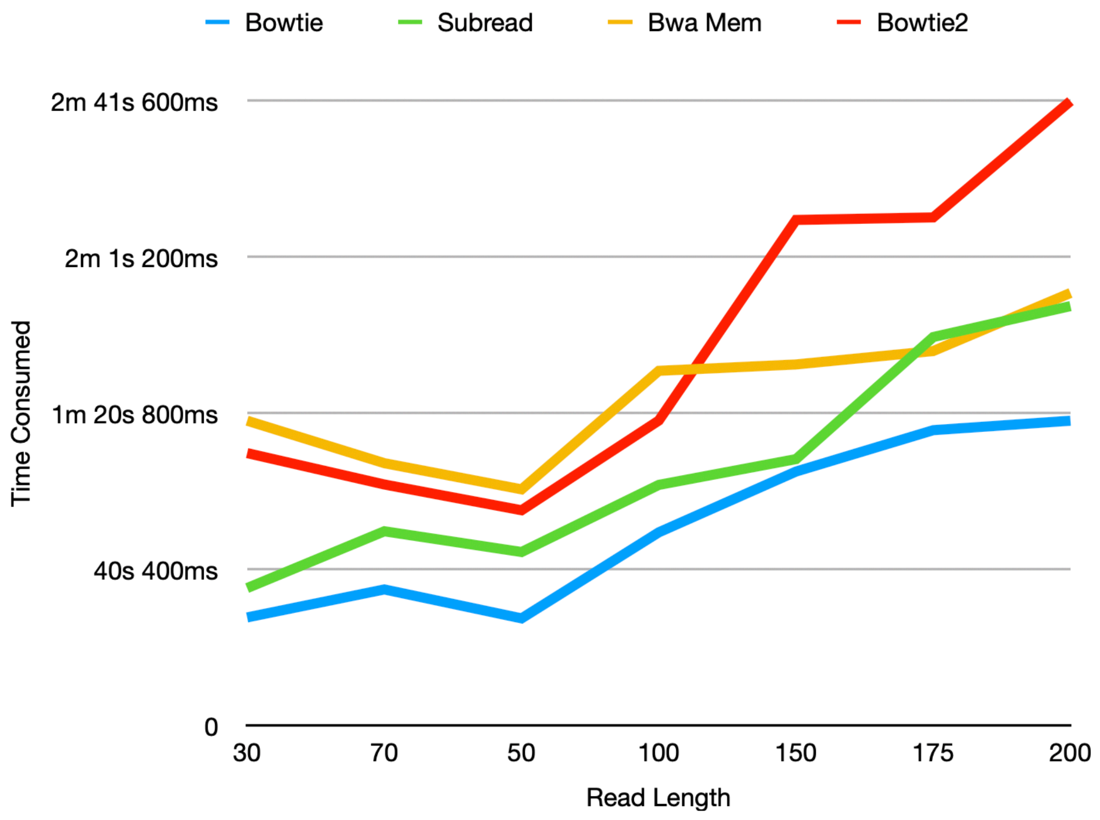


Figure 4: Time Consumed

9.3 Precision & Recall

9.3.1 Evaluation directly using Samtools

The algorithm is the same as before, the data is listed as the following table.

	BWA MEM	Bowtie2	Bowtie	Subread
Len 30 Precision	0.9933	0.9710	1.0000	0.9972
Len 30 Recall	0.9374	0.9778	0.9512	0.8429
Len 50 Precision	0.9808	0.9836	1.0000	0.9610
Len 50 Recall	0.9989	0.9718	0.9182	0.9217
Len 70 Precision	0.9889	0.9978	1.0000	0.9928
Len 70 Recall	1.0000	0.9955	0.8426	0.9147
Len 100 Precision	0.9342	0.9994	1.0000	1.0000
Len 100 Recall	1.0000	0.9946	0.6856	0.8531
Len 150 Precision	0.8781	1.0000	1.0000	1.0000
Len 150 Recall	1.0000	0.9994	0.3920	0.6699
Len 175 Precision	0.8608	1.0000	1.0000	1.0000
Len 175 Recall	1.0000	0.9998	0.2506	0.8717
Len 200 Precision	0.8467	1.0000	1.0000	1.0000
Len 200 Recall	1.0000	0.9999	0.1583	0.9003

9.3.2 Evaluation of sensitivity using Python code

The python code SamReader.py, which is given before, generate output that can precisely tell me: among those alignments, how many of them are correct. So we collect the data of each software when the read length is 70, and the code yields the following results:

	Correct Align	Incorrect Align	Total Align	Unaligned	Total
BWA-MEM	1999657	307	1999964	39	2000003
Subread	1792731	36632	1829363	170637	2000000
Bowtie2	1977050	1602	1978652	21348	2000000
Bowtie	1685820	0	1685820	314180	2000000

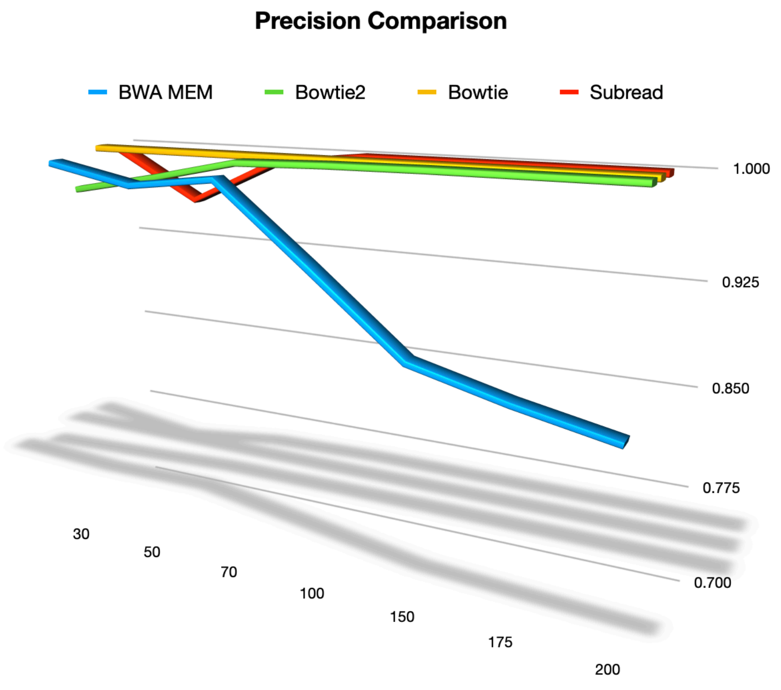


Figure 5: Precision Comparison

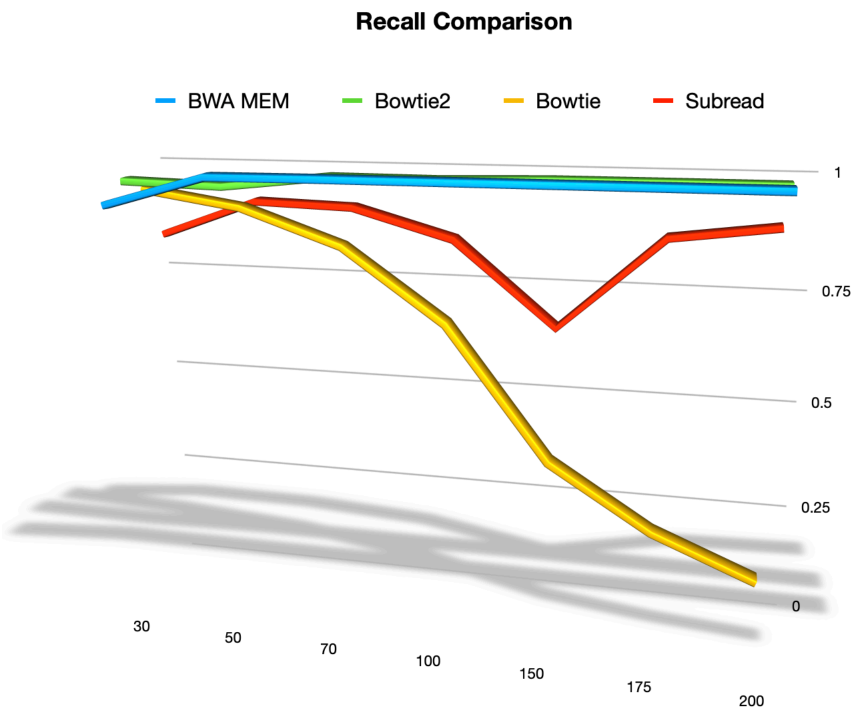


Figure 6: Recall Comparison

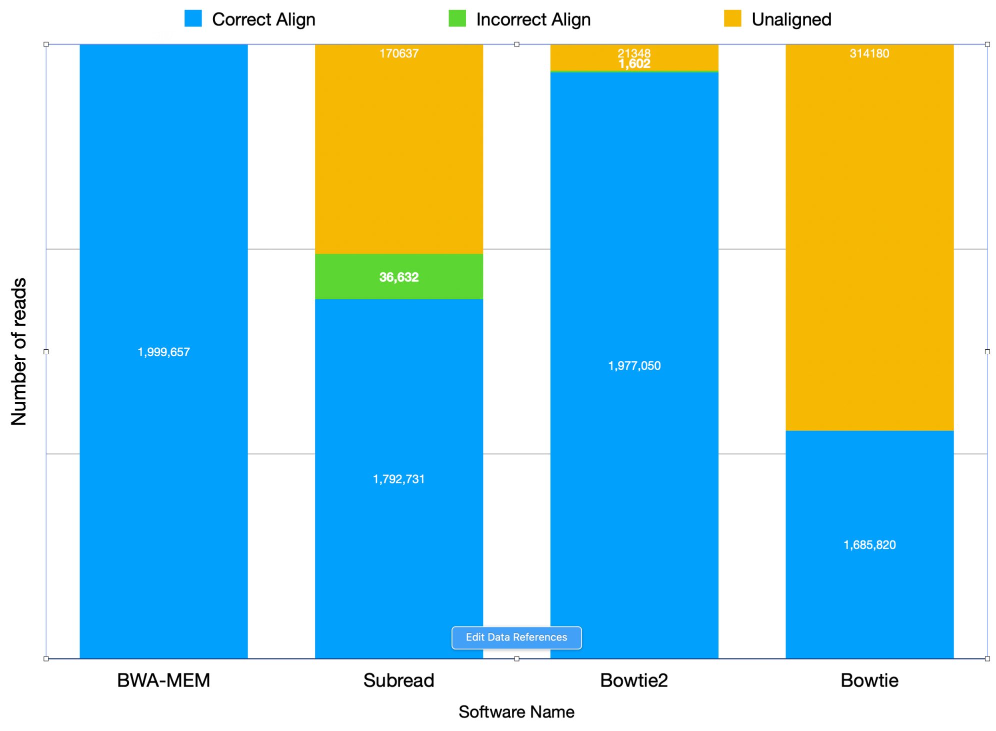


Figure 7: Performance evaluation using Python code to inspect sensitivity

10 Conclusion

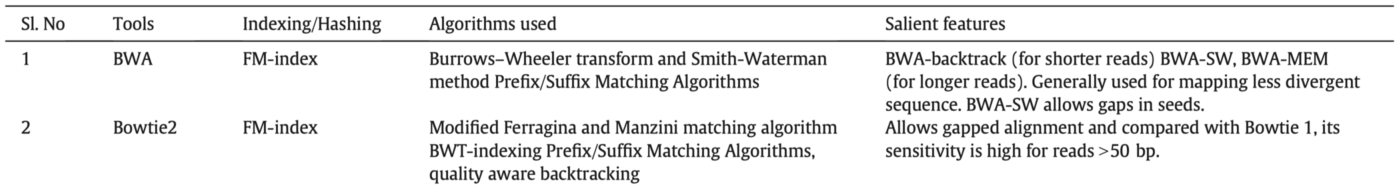
10.1 Bowtie Vs. Bowtie2

- Bowtie 2 fully supports gapped alignment with affine gap penalties. Number of gaps and gap lengths are not restricted, except via the user-supplied scoring scheme. Bowtie 1 only finds ungapped alignments. So the error tolerance of bowtie is way less than bowtie 2.[5]
- For reads longer than about 50 bp Bowtie 2 is generally faster, more sensitive, and uses less memory than Bowtie 1. The results of bowtie is horrible if reads get somewhat big. For relatively short reads (e.g. less than 50 bp) Bowtie 1 is sometimes faster and/or more sensitive.
- There is no upper limit on read length in Bowtie 2. Bowtie 1 had an upper limit of around 1000 bp.
- Bowtie 2's paired-end alignment mode is more flexible than Bowtie 1's. For example, for pairs that do not align in a paired fashion, it will attempt to find unpaired alignments for each mate.

Recomendation: Bowtie 1 was released in 2009 and was geared toward aligning the relatively short type of sequencing reads (up to 50 bp) prevalent at the time[5]. If your reads are shorter than 50 bp, you might want to try both Bowtie 1 and Bowtie 2 and see which gives better results in terms of speed and sensitivity. In my experiments, Bowtie 2 is significantly superior to Bowtie 1 for reads longer than 50 bp. For reads shorter than 50 bp, Bowtie 1 is extremely fast, but its sensitivity is not as good as bowtie 2.

10.2 Bwa Vs. Bowtie2

Basic feature comparison:



Sl. No	Tools	Indexing/Hashing	Algorithms used	Salient features
1	BWA	FM-index	Burrows–Wheeler transform and Smith-Waterman method Prefix/Suffix Matching Algorithms	BWA-backtrack (for shorter reads) BWA-SW, BWA-MEM (for longer reads). Generally used for mapping less divergent sequence. BWA-SW allows gaps in seeds.
2	Bowtie2	FM-index	Modified Ferragina and Manzini matching algorithm BWT-indexing Prefix/Suffix Matching Algorithms, quality aware backtracking	Allows gapped alignment and compared with Bowtie 1, its sensitivity is high for reads >50 bp.

Figure 8: Basic function comparison of Bowtie 2 and Bwa[7]

- Bowtie 2 and Bwa both exhibit very reliable recall and precision during our experiments. Normally, Bowtie 2 is slightly faster, but BWA is slightly more sensitive.
- In special case when read length is 150, the results of bowtie 2 is highly favorble, but the time it takes is nearly two times as bwa.
- Bowtie 2 has a very stringent default parameter on the length between two reads when it is in the paired-end mode. It needs repetitive tunning to find the maximum value unless you choose to set it to be very large to sacrifice some speed.
- For read length that is larger than 100, the precision of bwa begin to fall. In other words, it might align a bunch of reads that should not be aligned. However, bowtie 2 still remains to be highly precise.

Recommendation: If you encountered with a newly sequenced data, the first thing you must know is its insert length. Whether it is above 500 or not. Only if you have this information can you fully trust the results of bowtie2 and retain its high functionality. Also you might bother tune -X for a while which is quite cumbersome. However, bwa's default setting is not that stringent and can give you promising results in most cases.

Despite the facts above, their usage varies with read length. If your read length is below 70, you can try to use both. Bwa is slightly more sensitive and bowtie 2 is slightly faster. However, if your read length is 100, you should choose bwa, it has fairly high recall and speed, and its precision doesn't decrease much. This is verified by my experiments and Brittney N. Keel's[8]. For reads of length larger than 150, if you care about precision and recall, you should choose bowtie 2. If you care about speed and recall, you should choose bwa.

10.3 Subread

BWA and Bowtie2 demonstrated greater robustness than Subread, of nearly all read lengths. However, subread can feed you with the fastest alignment results in many situations. What's more, it is the most memory efficient choice on many occasions. If you are considering to construct a tool that requires very high speed, and not so stringent on precision and recall, subread might be your choice.

11 Files

The files attached are listed as follows

- auto.sh - The bash script that automate the whole simulation process
  - Readme.md - The manual file for auto.sh
  - file_handler.py - The python script that provides auxiliary function for auto.sh
  - Example_Report_file a list of file records the report of my experiment data, you can access them by using "cat report_1"
- bad_data_generator.py - An alternative approach of generating bad data.
- SamReader.py - The python code that handles the sam file to judge the correctness of aligned reads.

References

[1] H. Li, "Wgsim." https://github.com/lh3/wgsim.

[2] H. Li and R. Durbin, "Fast and accurate short read alignment with Burrows–Wheeler transform," *Bioinformatics*, vol. 25, pp. 1754–1760, 05 2009.

[3] "Manual reference pages - bwa." http://bio-bwa.sourceforge.net/bwa.shtml, Mar 2013. Accessed on 2020-03-24.

[4] "Online regex debugger." https://regex101.com. Accessed on 2020-03-24.

[5] P. M. Langmead B, Trapnell C, "Bowtie: An ultrafast, memory-efficient short read aligner." http://bowtie-bio.sourceforge.net/index.shtml. Accessed on 2020-03-27.

[6] Y. Liao, G. K. Smyth, and W. Shi, "The Subread aligner: fast, accurate and scalable read mapping by seed-and-vote," *Nucleic Acids Research*, vol. 41, pp. e108–e108, 04 2013.

[7] S. Thankaswamy-Kosalai, P. Sen, and I. Nookaew, "Evaluation and assessment of read-mapping by multiple next-generation sequencing aligners based on genome-wide characteristics," *Genomics*, vol. 109, no. 3, pp. 186 – 191, 2017.

[8] B. N. Keel and W. M. Snelling, "Comparison of burrows-wheeler transform-based mapping algorithms used in high-throughput whole-genome sequencing: Application to illumina data for livestock genomes1," *Frontiers in Genetics*, vol. 9, p. 35, 2018.