Project – Preprocessor for high throughput sequencing reads

Figure 1 The Illumina HiSeq (left) and ABI Ion S5 (right) are modern high throughput DNA sequencers.

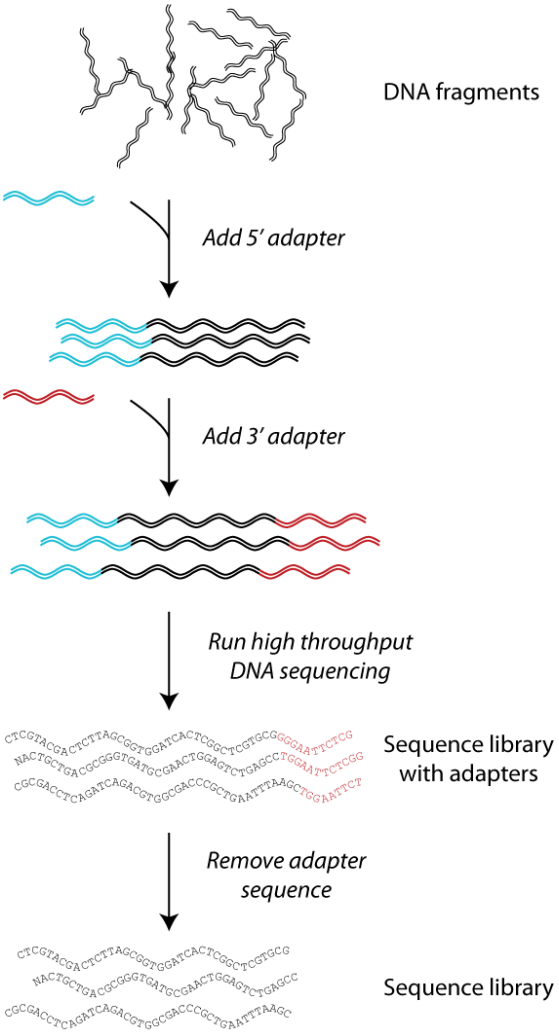

Figure 2 Constructing a sequence library from DNA fragments.

Modern high throughput DNA sequencing machines, such as Illumina's HiSeq and ABI's Ion S5 (Figure 1), can generate ~10^{10} nucleotides of data per day. Both technologies are based on massive parallel sequencing where millions of short (35-250 nucleotides) DNA fragments are sequenced in parallel. The end result is a large set S (sequence library), $|S| \approx 10^8$, of short, fixed-length DNA strings s_i, $\forall i,j\ s_i \in S, s_j \in S, |s_i| = |s_j|, |s_i| \in \{35, 50, 75, 100, 125, 250\}$.

Because of the technology used to sequence the DNA fragments, the sequences in the sequence library typically contain different suffixes that are prefixes to a specific DNA sequence (see Figure 2). This DNA sequence is called an adapter sequence and the suffixes corresponding to prefixes of this adapter sequence are called adapter fragments. The adapter fragments are artificially added sequences that do not correspond to any "natural" DNA. Consequently, the first step in analyzing a high-throughput sequencing library is to remove these adapter fragments. The goal of this project is to develop such a preprocessor.

Task 1 – Perfectly matching adapter fragments

The file `s_3_sequence_1M.txt.gz` contains a set S of sequences (note: one sequence per line; i.e. sequences are newline '\n' separated) from a high throughput sequencing experiment that used the following 3' adapter sequence: *a* = "`TGGAATTCTCGGGTGCCAAGGAACTCCAGTCACACAGTGATCTCGTATGCCGTCTTCTGCTTG`". Develop an algorithm that identifies all the sequences in S that contain suffixes that perfectly match a prefix of *a*. How many such sequences do you find? What is the length distribution of the sequences that remain after you have removed these perfectly matching adapter fragments? What is the asymptotic (and practical) running time of your algorithm?

Task 2 – Imperfectly matching adapter fragments

Because of sequencing errors, not all suffixes will perfectly match the adapter prefix, but contain one or several mismatches to the adapter. Develop an algorithm that identifies all the sequences in S that contain suffixes that match a prefix of *a* and where this suffix can contain up to a given percentage of mismatches to the prefix of *a*. How many such sequences do you find if you apply your algorithm to S and *a* from Task 1, given that the maximum percentage of mismatches is 10%? What is the length distribution of the sequences that remain after you have removed these imperfectly matching adapter fragments? What are the answers to the previous two questions if you set the maximum percentage of mismatches to 25%? What is the answer to the previous three questions if you allow insertions and deletions? What is the asymptotic (and practical) running time of your algorithm?

Task 3 – Sequencing errors and error distributions

Based on results from Task 2, can you estimate the rate of sequencing errors per sequence and per nucleotide? Are sequencing errors uniformly distributed across the sequences (i.e. is the frequency of sequencing errors the same at the start of the sequence as at the end of the sequence, or anywhere else)?

Task 4 – Finding the adapter sequence

For some datasets the actual adapter sequence could be unknown. However, the adapter sequence could still potentially be inferred by identifying frequently occurring suffixes within the sequence set S. Develop an algorithm that given a sequence set S, identifies the most likely adapter sequence *a* and use this algorithm to analyze the set found in the file `tdt4287-unknown-adapter.txt.gz`. What is the most likely adapter sequence? What is the length distribution of the sequences that remain after you have removed these adapter fragments? What is the running time of your algorithm? Does the set contain any highly frequent sequences; i.e. what is the frequency distribution of unique sequences in the set? Does the unique set contain any other common (proper) suffix patterns? Such additional common suffixes could indicate bias in the sequencing experiment. Does the set in `s_3_sequence_1M.txt.gz` contain additional common suffix patterns? What sequence does your algorithm return if you use your algorithm to analyze the files `s_3_sequence_1M.txt.gz` and `Seqset3.txt.gz`?

Task 5 – De-multiplex barcoded library

A common strategy for reducing sequencing costs when sequencing multiple samples is to use a specific 3' adapter sequence (barcode; see Figure 3 for an example) per

sample, mix all sample libraries and run a single sequencing reaction, and then use the sample-specific adapter sequence (barcode) to identify (de-multiplex) which sample any given sequence belongs to. The file `MultiplexedSamples.gz` contains the results of such a multiplex sequencing experiment. Your tasks are (1) to identify the barcodes (3' adapters) used and thereby how many samples were multiplexed, (2) identify how many sequences that were sequenced from each sample, and (3) identify the sequence length distribution within each sample. What is the most frequently occurring sequence within each sample?

HINT: All barcodes have the same length. Moreover, most barcodes in use are fairly short (4-8 characters).

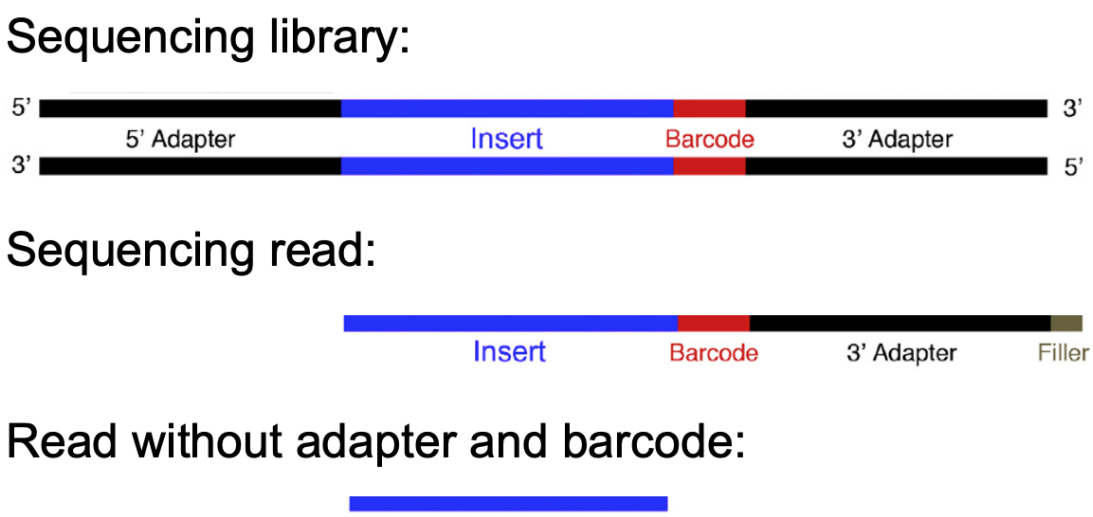

Figure 3 Example of a sequencing library with barcodes.

Deliverables

1. Project report: Your project report should contain a short **Introduction**, a **Method** section that explains the methods you have developed to solve the different tasks, and a **Result and Discussion** section that presents your results. In the Method section, you should use pseudo code to explain your algorithms and references to algorithms from the curriculum when appropriate. In the Result and Discussion section you should use graphs to present your results and discuss these when appropriate.
2. Oral presentation: Prepare an 8 minutes presentation of your methods and results. You should aim for 2 slides per task. Refer to algorithms from the curriculum if you have used those for your solutions; otherwise, you should explain your own solutions.