Apollo: a sequencing-technology-independent, scalable and accurate assembly polishing algorithm.

Abstract: **Motivation:** Although third generation sequencing technologies have the ability of sequencing long reads but have large amount of errors so it can have incorrect reads. Because of these errors we used Assembly polishing algorithms to fix these errors by using information from alignments between reads and the assembly but it polish small assembly or certain sequencing technology. **Result:** Because of this problems**,** we use apollo. It's a universal assembly polishing algorithm and accurate to polish an assembly of any size. Our aim of using it is to provide a single algorithm as the only algorithm that can polish large algorithm and uses read sets from all available sequencing technologies.

Introduction: High-throughput sequencing are used to generate a huge value of sequencing data at low cost. it have two significant limitations, the first one can only sequence fragments of the genome. this lead to reconstruct the original full sequence. the second one is that they introduce non-negligible insertion, deletion and substitution errors into reads. It depends on the method for reconstructing the original sequence. HTS is divided into two types: second and third generation sequencing technologies. Second generation sequencing technologies generate the most accurate reads but the length of the read is short. This introduces challenges in both read alignment and de novo genome assembly. In read alignment, a short read can align to multiple locations in a reference well. It should be found a high computational complexity in de nova to identify overlaps between reads. Molecule Real Time and Oxford Nanopore Technologies are able to produce long reads. PacBio reads can have more insertion errors than other error types while insertion errors are the least common errors for ONT reads. Existing solutions that try to solve the problem of errors assemblies when using de novo genome assembly can be divided to two types. First: a typical solution is to correct the errors of long reads by using high coverage reads from the same sequencing technology or additional reads from more reliable second-generation sequencing technologies. There are several available error correction algorithms that use additional reads to locate and correct errors in long reads, LoRDEC, LSC, and LoRMA Second: method for removing errors in an assembly is called assembly polishing, Using the alignments of either long or short reads to the assembly, an assembly polishing method helps to correct the assembly's errors. There are a variety of assembly polishing algorithms that use different methods for detecting and changing dissimilarities in the assembly like Racon, Quiver and Pilon. Many of these assembly polishing algorithms have the drawback of only working for reads from a small number of sequencing technologies. For two reasons, there are scalability issues of using polishing algorithms to polish a broad genome and, as a result, running assembly polishing algorithms several times. first: None of the polishing algorithms can handle massive genomes in a single run because they demand a lot of computing power. Second: splitting a large genome into smaller contigs and running polishing algorithms several times necessitates additional work to compile and combine the multiple findings into a polished large genome assembly. Apollo's machine learning algorithm is based on two key steps: training and decoding. We compare Apollo with Nano polish, Racon, Quiver and Pilon using datasets that are sequenced with different technologies. We first demonstrate that Apollo scales better than other polishing algorithms in polishing assemblies of large genomes using moderate and high coverage readings, using datasets from various sequencing technologies. Apollo is the only algorithm that can use reads from multiple sequencing technologies in a hybrid manner.

Related Work: Different third-generation sequencing technologies result in different error profiles, There are two kinds of existing solutions for overcoming the issue of error prone assemblies by using de novo genome assembly. First, a typical solution is to correct the errors of long reads. Errors are corrected by using high coverage reads. The second method for removing errors in an assembly is called assembly polishing, An assembly polishing process attempts to correct the errors of the assembly using the alignments of either long or short reads to the assembly. There are various assembly polishing algorithms that use various methods for discovering dissimilarities and modifying the assembly like Racon, Pilon. there are scalability problems associated with using polishing algorithms to polish a large genome and, therefore, running assembly polishing algorithms multiple times for two reasons, these assembly polishing algorithms cannot polish large genomes in a single run if the available computational resources are not tremendous. Apollo, that corrects errors in an assembly. when we compare Apollo to other competing algorithms, these experiments show that Apollo usually produces assemblies of similar accuracy to competing algorithms: Nano polish, Pilon, Racon and Quiver. Apollo produces assemblies with less accuracy than that of Racon and Quiver. These experiments are based on ground truth k-mer similarity calculation between an Illumina set of reads and a polished assembly. These comparisons show that Apollo can polish an assembly using reads from any sequencing technology while still generating an assembly with accuracy usually comparable to the competing algorithms.

Materials and methods:

Apollo creates, trains, and decodes a hidden Markov model with a profile. To polish an assembly, use a graph (pHMM-graph). As illustrated in Figure 1, Apollo polishes assemblies utilising two input preparation processes that are external to Apollo (pre-processing) and three internal procedures. External tools such as an assembler and an aligner are used in the first two preprocessing processes to produce Apollo inputs. An assembler generates assembly contigs by first using reads .Second, to create read-to-assembly alignment, an aligner aligns the reads utilised in the first step, as well as any extra reads from the same sample, to the contigs. Apollo then constructs a pHMM-graph for each contig using the assembly created in the first phase. To account for all conceivable error kinds, a pHMM-graph had states, transitions between states, and probabilities associated with both states and transitions. Insertion, deletion, and substitution mistakes, as well as chimeric mistakes, are examples of faults that a sequencing method might incorporate into a read. As a result, these mistakes can be corrected by removing, inserting, or swapping the matching base pair, as appropriate. Apollo finds a route in the pHMM-graph that excludes the states that cause the contig to be incorrect. Fourth, Apollo updates using the read-to-assembly alignment. The Forward–Backward approach changes the prior probability of the graph based on the similarity between a read and the aligned area in the assembly during training. Fifth, Apollo uses the Viterbi method to discover the path with the lowest error probability in the pHMM-graph, which corresponds to the polished version of the relevant contig.

Results:

Experimental setup: We implemented Apollo in C++ using the Seq An library. Our evaluation criteria include three different methods to assess the quality of the assemblies. First, we use the provided dnadiff tool. To calculate the accuracy of polished assemblies by comparing them to highly accurate reference genomes. Second, we compute the k-mer similarity between filtered Illumina reads and an assembly using

sourmash. Third, we employ QUAST to provide an additional assembly quality evaluation based on the mapping of filtered Illumina reads to assemblies. Both k-mer similarity and QUAST allow you to evaluate assemblies without having to use a reference. Based on our evaluation criteria, we compare Apollo to four state-of-the-art assembly polishing algorithms: Nanopolish, Racon, Quiver and Pilon. We use state-of-the-art tools to construct an assembly and to generate a read-to-assembly alignment before running Apollo, which correspond to the input preparation steps. After assembly generation, we divide the long reads into smaller chunks of size 1000 bp. For Apollo, dividing long reads into chunks prevents possible memory overflows due to the memory-demanding calculation of the Forward–Backward algorithm. Even though it is still possible to use long reads without chunking.

Datasets: we use DNA-seq datasets from five different samples sequenced by multiple sequencing technologies. We use a dataset from a large genome to demonstrate the scalability of polishing algorithms. We use the E.coli O15 , E.coli O157:H7, E.coli K-12 MG1655 and Yeast S288C datasets to evaluate the polishing accuracy of Apollo and other state-of-the-art polishing algorithms in four way. First, we evaluate whether using a hybrid set of reads with Apollo results . Second, we measure the performance of the polishing algorithms when they polish the assemblies only once. Third, we subsample the E.coli O157 and E.coli K-12 datasets into 30 coverage to compare the performance of algorithms when long read coverage is moderate. Fourth, we additionally use the Human HG002 dataset to measure the k-mer distance and quality assessment of the assemblies using sourmash and QUAST, respectively.

Applicability of polishing algorithms to large genomes: We use the polishing algorithms to polish a large genome assembly to observe whether the polishing algorithms can polish these large assemblies without exceeding the limitations of the computational resources. we use the PacBio and Illumina reads from the human genome sample of the Ashkenazim trio (HG002, Son) to polish a finished assembly of the same Ashkenazim trio sample. we make four key observations. First, Pilon, Quiver and Racon cannot polish the assembly using the whole sets of PacBio and Illumina reads due to high computational resources. Racon and Pilon exceed the memory limitations while using either the PacBio or Illumina reads to polish the human genome assembly. Racon and Quiver can polish the large genome using a low coverage set of PacBio reads.

Polishing accuracy: We first investigate whether employing a hybrid collection of reads in a single polishing run is preferable to polishing an assembly twice using data from a single sequencing method. Second, we evaluate assembly polishing algorithms and compare them to each other. When polishing a Canu-generated assembly with only a high coverage set of PacBio or Illumina reads, Apollo outperforms Pilon and is equivalent to Racon and Quiver. Apollo performs better than Pilon and Nanopolish when polishing a Miniasm-generated assembly using only a set of Illumina and ONT reads, respectively. Apollo's polishing score differs only by at most 1:21%.

Reference-independent quality assessment: We filter Illumina reads in three steps to get rid of erroneous short reads before using them. First, we remove the adapter sequences , Second, we apply contaminant filtering, Third, we map the reads generated after the first three steps to the reference and filter out the reads that do not map to the reference. In k-mer similarity calculations, Jaccard similarity provides how a set of k-mers of both Illumina reads and an assembly are similar to each other. We offer five significant observations based on the k-mer similarity data between Illumina readings and human genome assemblies. We use QUAST, a quality assessment tool for genome assemblies.