Genome Project: Final Report

Predictive model and Paired End based Novel Genome Assembler

Advised by Prof. Poonam Goyal

Satwik Bhattamishra
f2014319@pilani.bits-pilani.ac.in

Chinmay Deshmukh
f2014770@pilani.bits-pilani.ac.in

I. INTRODUCTION

In this project, we present a novel approach for the genome assembly problem to generate high quality contigs as well as preserve the efficiency of the assembler. Genome assembly has been an active area of research in the recent times and there have been several distinct approaches that have been proposed in the past few years. Yet a significant number of assemblers could be classified under De-Bruijn graph methods since they are fundamentally developed based on the de-bruijn graph approach and try to optimize as mush as possible in terms of time and space. Although de-bruijn graph approaches have been shown to produce sequences of good quality and are efficient [3] [8], however they have some irremediable limitations such as the loss of paired-end information, unresolved branches and the storage limitation of the graph itself. Methods have been proposed to create unitigs and omnitigs [6] to reduce the computation of the de-bruijn graph. Other notable approaches include PERGA [9], which uses the paired-end information and predictive modeling for base-by-base extension without using de-bruijn graph. However, it still has some limitations and we will discuss more on this issue later in this paper. Here we propose an approach that exploits the efficiency of de-bruijn based approaches and also uses paired-end information and predictive modeling to generate contigs of higher quality.

II. RELEVANT PRIOR WORK

The most popular approaches for genome assembler such as Minia [3] and Velvet [8] are based on de-bruijn graph in which an overlap graph is constructed based on a fixed size k-mer and the contigs are constructed based on traversal of these graphs. As mentioned earlier it has been established that direct use of paired-end information in-case of complete de-bruijn graph is not feasible. De-Bruijn graph method was developed without keeping in mind the paired-end data since paired-end information is a part of relatively new technology of sequencing machines. Assemblers such as Meraculous [1] rely on traversal of the subgraph so as to efficiently use the paired-end information. Other than that, recently there have been methods developed to generate unitigs first before using the de-bruijn graph approach. In BCALM[2], they first provide an efficient way of creating unitigs which are then fed into Minia for the final assembly of contigs. For further discussions unitigs can be considered as safe[1] strings of maximal length. In sparse assembler[7], inspired by the unitig approach, they show a method to reduce storage of unnecessary nodes in the de-bruijn graph by skipping a set of k-mers while building the k-mer overlap graph.

[1]Safe here means that during the extension only one possible option is encountered in each state

Generally in case of de-bruijn based methods, the extension of the contigs stop when branches[2] are encountered. PERGA[9] proposed an alternate approach without using de-bruijn graph and using paired end information and machine learning for the extension of sequences. In case of PERGA the contigs are extended base-by-base based on read alignments. In each case there are three scenarios based on the alignment.

1. If paired-end and single-end alignment indicates only one base for extension, extend with that base.
2. If the possible extensions based on alignment are ambiguous, use SVM to predict the next base based on 4 features.
3. If the predictive model is not confident for any specific direction, use look-ahead approach. If even that fails, then stop.

PERGA uses the above rules for extension of each base to produce a contig. The branches are resolved using the predictive model unlike de-bruijn graph approaches. However, one can see that alignment of reads for each extension is computationally expensive. Here we are interested in developing a method that is able to generate high quality contigs using paired end information and predictive modeling but avoids the excessive computation cost of read alignment for every extension.

III. PROPOSED METHOD

In this section we explain our approach to generate contigs efficiently based on paired-end information and predictive modeling. There are two major motivations to develop this approach. Firstly, although de-bruijn graphs methods are efficient, they may fail to resolve branches and are unable to properly use the paired-end information in the reads data. Using these information can allow us to generate contigs of relative higher quality and thus a better assembly. Secondly, methods like PERGA use read alignment for every single extension and although it is good at resolving branches, scenarios such as unitigs where we are already confident that the extension is safe leads to unnecessarily expensive computation due to read alignment.

A. ASSEMBLER DETAILS

Thus keeping in mind the above points, we present the following method. We generate a set of k-mers from the given read data and store it in a hash table. Along with the k-mer is a set of read ids from which the k-mer could be generated. We traverse the table a fill the neighbor variable which denotes the possible extensions ($\{A, T, G, C\}$) for each k-mer in each direction (5'-3' and 3'-5'). While creating the contigs, we store the read ids that were used to create the contigs which will be later used for the alignment process. To start the extension process we randomly select a k-mer and check the neighbor variable. If only one neighbor exists then we extend the contig and add the read ids to the set in the contig. Until the case where more than one neighbor exist, the generated string could be considered as a unitig or a safe string. If more than one neighbor exist, then we pause the extension process and extend the strings in the possible routes separately in the same way upto N bases as if they were a contig. If they converge at some point then it is case of bubble and they can be resolved using tourbus algorithm similar to the approach followed by

[2]Specifically branches here mean excluding bubbles and tips

velvet[8]. If one of the branches is found to be a very short string with no possible extension after some point then it is a case of tip and it will be discarded. Lastly and most importantly, in the case where it is not a bubble or a tip, we align the relevant reads with contig and branches to generate a set of statistical features. Based on those features we will use a machine learning model to predict the correct path. If there are paired-end reads among the aligned reads, then we will also use them in the decision process based on the insert size of the paired ends. Finally we will use a linear combination of the predictive model and the paired end information to score each route and if the score is above a threshold then we will take that path. If we are unable to resolve the branch based on these information then we will break the extension and return the generated string as our contig.

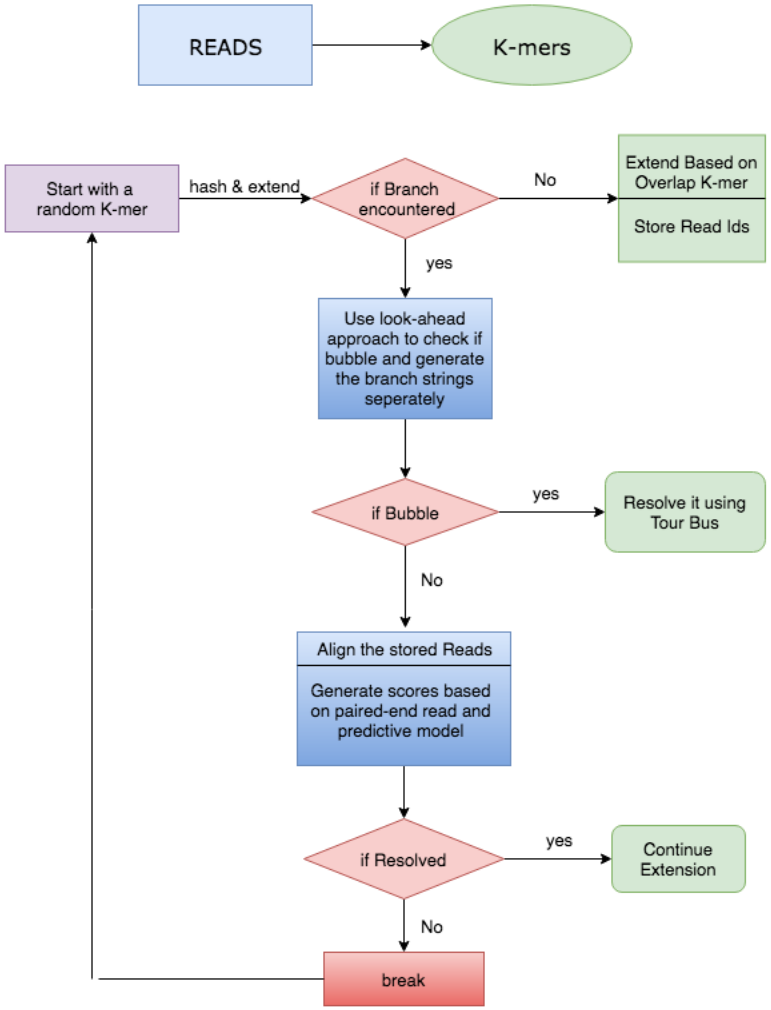

Figure 1: Overall design of the proposed assembler

Figure 1 shows the overall schematics of our proposed assembler. It is important to note here

that unless we have determined that the given scenario is a branch and not a bubble or a tip, we are avoiding the alignment of reads thus saving unnecessary computations.

B. TRAINING THE PREDICTIVE MODEL

To train the predictive model we first need to generate appropriate data. We are going to run the assembler without the branch prediction module on a set of read datasets and whenever there is a scenario where branch resolution is required, we will generate contigs by going through each branch. Then for each generated contig we will score them based on quality using quast[4] and the features that are mentioned in Satwik's previous semester's project report. Thus for each branch we get a quantitative score along with a set of alignment features. We train the machine learning model with each of these pair of branch contig and quality score and optimize it based on cross-validation so as to get a final model that can predict the correct branch in our assembler.

C. EXAMPLE

To provide a better understanding, in this section, we are going to explain how our assembler would go through a specific set of reads to generate a contig.

Neighbor	READ ID	K-MER
0001	2,6	AATG
0101	2,6	ATGC
0000	2	TGCT
0010	1, 4, 6	TGCC
0100	1, 4	GCCG
1000	1	CCGT
0101	4	TTGC
0100	3	TACG
1000	3, 5	ACGT
0001	3	CGTA
0010	5	GTAC
0100	5	TACG

Figure 2: Reads dataset and generated k-mer hashtable

Figure 2 shows the table generated for a specific set of reads with k-mer size 4. We store the read ids along with the k-mer and the neighbor variable is one-hot encoded to indicate the possible extensions.

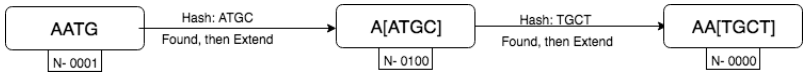

Figure 3: Normal extension, generated contig: AATGCT

Figure 3 shows the case where we start our extension with the first k-mer in the table and we do not encounter any branches along the way. Figure 4 shows the case where we encounter a branch and we extend each branch separately as if they were contigs. As mentioned earlier we extend each of them upto N bases to determine if they are bubbles or tips. If they are neither of those then we will attempt to resolve them using machine learning and paired information.

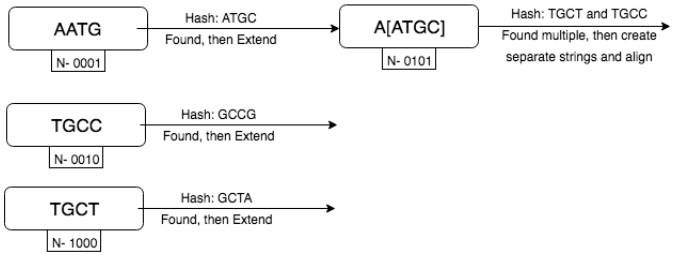

Figure 4: Branch Scenario, separate extension

IV. IMPLEMENTATION AND RESULTS

The above mentioned design is implemented in C++. The unitigs generated by our method work correctly based on the results of quast. The implementation of unitigs were tested on 5 different reads dataset and the results generated were fairly similar (better in most cases since bcalm is optimized) to the ones generated by bcalm.

Genome statistics	unitigs_60x
Genome fraction (%)	97.985
Duplication ratio	2.117
Largest alignment	3846
Total aligned length	9 622 589
NGA50	996
LGA50	1693

Figure 5: Statistics of unitig assembly

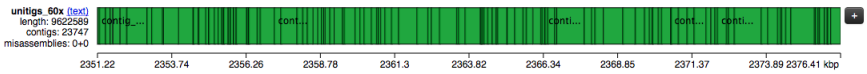

Figure 6: Alignment of unitigs

The unitigs were created by joining k-mers which had only one neighbor. Once the extension process finds multiple possible extensions, the unitig extension stop and we store the pointers to the neighboring unitigs that were created from the multiple possible extensions. These neighboring pointers are later used resolve the branches when forming a contig.

For the creation of contigs we initially start with a unitig and check its possible extensions (other unitigs) and based on common reads in the bridges, paired-end reads and predictive model we decide which possible extension is to be chosen. The initial unitig is chosen based on the amount of reads that were used to form the unitig. This method prevents starting from a unitig that could be a part of a bubble and possibly be generated based on sequencing error. Starting from a unitig which is a part of a bubble results in additional computation to traverse back and check the right unitig and thus the heuristic results in partly avoiding such computations.

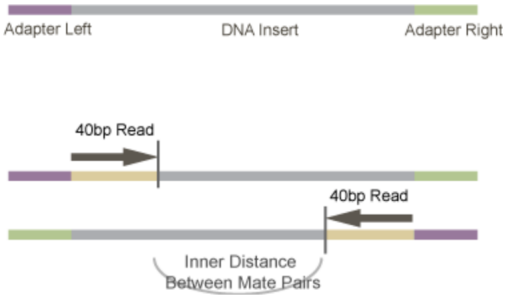

Figure 7: Paired End Reads

During the selection of multiple extensions, the first way to resolve the branches is to check the number of aligned reads in the bridges among multiple reads. Since initially we start with a unitig, the length of the whole contig is relatively low and this prevents us from using paired-end reads and predictive model. Since most paired-end reads sequenced using illumina have an insertion length of about 500 base pairs, unless the length of the whole contigs is greater than 500 base pairs we cannot align the mate pairs in the primary contig and its extension. Once our given contig has a length over 500 base pairs, we check if a mate pair can be aligned with the primary contig and any of its possible extension. If paired end reads are found to be align then it gives us high confidence for that extension. For the read files generated from illumina sequencing machines, two consecutive reads are mate pairs (reads 1 and 2, 315 and 316). Thus for each read id aligned with out primary contig we check if its next read (if read id is odd) or its previous read id (if read id is even) is present in an extension.

This method cannot be directly used in de-bruijn graph methods and assemblers such as platanus [5] remap the bubbles and check if paired-end alignments are possible. This is especially important in case of heterozygous genomes which different alleles resulting in different k-mers for the same part of homologous chromosomes. The heterozygosity results in multiple peaks (refer Figure 8) in the distribution of the k-mer count as opposed to a single peak in case homozygous genomes. Thus for a same region different k-mers occur and result in complex bubbles which are not efficiently resolved using de-bruijn graph approach. Assemblers such as platanus have found the use paired-end reads to be beneficial in case of heterozygous genomes.

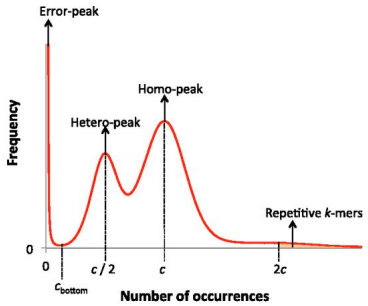

Figure 8: K-mer count distribution

Thus for each possible extension we keep the count of the paired-end reads that are aligned and along with that we predict the quality of each extension using a neural network. To train the network or any other model we first generate appropriate data. With our standard assembler (without model), whenever a branch is encountered we create a row of data in which the first column contains the string of the primary contig which is being extended. The following column contains the list of tab separated read ids which were used to form the contig. The next 16 columns contain a pair of possible extension starting with each k-mer along with its read id list [(1+1) $\times$ 4] and for each direction (8 $\times$ 2 =16). Thus we generate a csv file in which each row contains the contig which is being extended and its possible extensions (along with read ids). In this way from the synthetic 60x coverage reads dataset we generated about 80,000 data points and an additional 92,000 data points from the synthetic 100x coverage dataset of E. Coli.

For each row in our generated dataset, we concatenate the primary contig with each of its possible extension and generate its quality score using quast. We then convert the ordinal variable into a categorical one by dividing them into n bins. Based on our experimentation we have set n as 6. Once we have labeled each possible extension with a integral quality value we train our model using 5-fold cross-validation (3:1 split). One additional measure we have taken is to limit the overall contig size. When a primary contig being extended has a length greater than 5000 and its possible extensions have length less than 250, then the quality score is overshadowed by the properties of the primary contig making the possible extensions indistinguishable. Thus we limit the length of sequence we consider from the primary contig (one being extended) to 3 $\times$ maximum length of possible extensions. This helps the predictive model to distinguish between the different possible extension and reduces the influence of the primary contig in the overall decision process.

TABLE I: Scores of multiple models (%) (5-fold Cross Validation)

Method	F1 Score	M-Log Loss
Logistic Regression (Baseline)	96.298	0.103
Random Forest	97.104	0.0864
Xgboost	97.377	0.0811
Neural Net (2 layers)	97.163	0.0842

The models were implemented using python and we tested 4 different models to judge whether our neural network performed well. Logistic Regression is taken as baseline. Based on the results the gradient boosting model performs the best but the neural network is still subject to hyper-parameter tuning. Regardless of the performance of each model, since our network performs similar to gradient boosting machine and random forest, we move forward to use it for our branch resolution problem. The model trained using python is saved the tensorflow graph is retrieved using the c++ api and used in our assembler.

Currently we are using majority voting system to decide the correct branch. We combine the scores of the paired end read count and predictive model and choose the branch with the highest score. This can certainly be more finetuned to increase its performance. The results of our assembler on E. Coli dataset is shown in Figure 9 and the alignment is show in Figure 10. Although we

could get high NGA50 score and several long contigs, the maximum one being of length 138,269 which is significantly greater than the one generated by Minia. But our assembler generated a few mis-assemblies as shown in Figure 10. There could be two possible reasons for this behavior that we are investigating. The first could be some minor error in our implementation and the second could be the way we have chosen our branch. Currently, our condition for breaking an extension is very relaxed (since we are taking majority voting system) and having a more stringent condition could help us avoid such misassemblies at cost of a decrease in the length of the contigs.

Genome statistics	lop_contig_v7_kmer_tiprem_pai...
Genome fraction (%)	97.997
Duplication ratio	1.984
Largest alignment	138 269
Total aligned length	9 017 209
NGA50	42 125
LGA50	39

Figure 9: Statistics of contig assembly

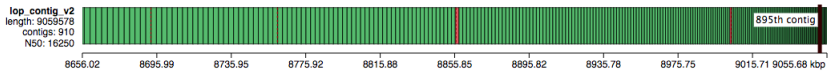

Figure 10: Alignment of contigs

We are still experimenting with our assembler and we also intend to include the results of our assembler on a diploid genome dataset. To the best of our knowledge, there is no assembler that uses paired end information and predictive model in the same way as mentioned above and consequently there is no parallel assembler that uses paired-end information and predictive model.

V. CONCLUSION

Thus in this paper, we described our design, implementation and results to generate high quality contigs using paired-end information and machine learning. We avoided excessive computations due to read alignment by following a k-mer based extension for safe sequences. In summary our design of the assembler has the following features:

- Efficient extension when alignment is not required (For safe sequences: unitigs)
- K-mer based extension for each base.
- Use of Paired End Information
- Branch Resolution using prediction for higher contig quality
- No De-Bruijn Graph

Our assembler resulted in contigs of higher length for multiple datasets although the occurrence of mis-assemblies is a common pattern in those experiments. We believe the mis-assemblies

could be due to relaxed conditions to break a contig extension and tuning the scoring/decision method could result in avoiding those mis-assemblies with a slight decrease in the average length of the contigs.

REFERENCES

[1] Jarrod A Chapman, Isaac Ho, Sirisha Sunkara, Shujun Luo, Gary P Schroth, and Daniel S Rokhsar. Meraculous: de novo genome assembly with short paired-end reads. *PloS one*, 6(8):e23501, 2011.

[2] Rayan Chikhi, Antoine Limasset, and Paul Medvedev. Compacting de bruijn graphs from sequencing data quickly and in low memory. *Bioinformatics*, 32(12):i201–i208, 2016.

[3] Rayan Chikhi, Guillaume Rizk, et al. Space-efficient and exact de bruijn graph representation based on a bloom filter. *Algorithms for Molecular Biology*, 8(22):1, 2013.

[4] Alexey Gurevich, Vladislav Saveliev, Nikolay Vyahhi, and Glenn Tesler. Quast: quality assessment tool for genome assemblies. *Bioinformatics*, 29(8):1072–1075, 2013.

[5] Rei Kajitani, Kouta Toshimoto, Hideki Noguchi, Atsushi Toyoda, Yoshitoshi Ogura, Miki Okuno, Mitsuru Yabana, Masayuki Harada, Eiji Nagayasu, Haruhiko Maruyama, et al. Efficient de novo assembly of highly heterozygous genomes from whole-genome shotgun short reads. *Genome research*, 24(8):1384–1395, 2014.

[6] Paul Medvedev. Modeling biological problems in computer science: A case study in genome assembly. *arXiv preprint arXiv:1706.05429*, 2017.

[7] Chengxi Ye, Zhanshan Sam Ma, Charles H Cannon, Mihai Pop, and W Yu Douglas. Exploiting sparseness in de novo genome assembly. In *BMC bioinformatics*, volume 13, page S1. BioMed Central, 2012.

[8] Daniel R Zerbino and Ewan Birney. Velvet: algorithms for de novo short read assembly using de bruijn graphs. *Genome research*, 18(5):821–829, 2008.

[9] Xiao Zhu, Henry CM Leung, Francis YL Chin, Siu Ming Yiu, Guangri Quan, Bo Liu, and Yadong Wang. Perga: a paired-end read guided de novo assembler for extending contigs using svm and look ahead approach. *PloS one*, 9(12):e114253, 2014.