FASTQ denoising for assembly

Shubham Chandak, Kedar Tatwawadi, Tsachy Weissman

Introduction

This document presents a summary of our attempts at denoising of FASTQ files for improvement of assembly. Most of the work was performed in Autumn quarter of 2017-18 (Sep-Dec 2017). The work involves short reads obtained using Illumina technology.

Two possible pipelines for use of FASTQ files are for variant calling and for assembly. The variant calling pipeline involves alignment to a reference genome followed by several calibration steps and finally variant calling (again followed by recalibration). On the other hand, assembly directly uses FASTQ files without requiring any reference. Even though many organisms have reference genomes available, assembly is still useful for organisms showing high variability between individuals [11]. Since denoising for variant calling can be performed after alignment, where the process becomes much simpler, we decided to pursue FASTQ denoising with improvement in assembly as the goal. Another possible goal could be to improve the speed and memory requirements of the assembler, but this was a secondary goal and the existing denoisers do not appear to achieve this [1].

In this context, [1] performed a survey of various existing denoisers, comparing their performance on assembly. Using the state-of-the-art assembler for small (bacterial) genomes (SPAdes, [2]), they found that even though the denoisers correct most of the errors, their impact on assembly is not as good. Many denoisers make the assembly worse, while some offer slight improvements. They explain this apparent anomaly by pointing out that the assemblers have significant built-in error correction capabilities, and a denoiser is useful only if it can handle the errors which the assembler cannot correct. Also, certain k-mer based denoisers tend to remove a lot of low coverage k-mers, thus making the assembly of those regions impossible.

Existing Denoisers

There are multiple categories of denoisers, including k-mer based and MSA (multiple sequence alignment denoisers). K-mer based denoisers count the frequencies of all the k-mers and then try to modify the low-frequency k-mers to high-coverage k-mers. Reckoner [5,6] is an example of this class. MSA methods try to align reads from the same genomic region to each other, form a consensus sequence from these and then denoise the reads according to the consensus. Karect [4] is such a denoiser. For each read, Karect searches for overlapping reads using k-mer based methods. During this process, Karect restricts the maximum number of reads found per position to reduce the effect of repetitive sequences on denoising. After this, Karect assigns weights to each base of each overlapping read depending on its quality and the mismatches of that read to the read being denoised. This allows Karect to utilize reads with up to 25% mismatch to the current read without letting them affect the quality of denoising. Many denoisers in both the classes use quality scores, either by converting them into weights or by using them as thresholds for denoising.

Among the existing denoisers, Karect seems to outperform the others, both in fraction of errors corrected and in assembly performance. The SPAdes assembler includes Bayeshammer [3] as a denoiser. Bayeshammer is a k-mer based method built for single cell sequencing data, which have large variations in coverage. Thus, it can handle low coverage regions much more successfully than other k-mer based denoisers. During evaluation, we also considered a read trimmer which trimmed reads based on quality scores (Sickle, [10]).

Assembly

[11] contains a detailed survey on assemblers. Here we will talk only about SPAdes [2] since it was used for our experiments. SPAdes is an assembler which was initially developed for single cell sequencing (SCS) data with highly non-uniform read coverage. SPAdes is a de Bruijn graph based assembler, but it tries to directly incorporate read pairing information into a paired de Bruijn graph. It combines information from multiple k-mer sizes to build a multisized assembly graph, allowing for different k values in low coverage and repeating regions. SPAdes also has extensive built-in error correction mechanism. It uses an iterative algorithm which tries to correct suspected erroneous k-mers one by one, while trying to make sure that the low-coverage k-mers are not lost in this process.

For evaluation of assemblies, we typically use the N50 and related scores. To calculate the N50 score, one first sorts the contigs in decreasing order of length. The N50 score the length of the contig such that the sum of lengths of contigs greater than that contig is 50% of the total length of all contigs. The NG50 score uses the reference genome length instead of the total length of all contigs. The NA50 score only includes contigs that align to the reference contig. The NGA50 score has both these features. For all the scores, higher is better. Even though these scores are not perfect indicators of assembly quality, these are typically used for evaluation of assemblies.

The choice of assembler can have a huge impact on the results of a denoiser. For example, [1] observes that most denoisers improve assembly significantly on Velvet [12], but the NGA50 scores after denoising are much lower than the corresponding scores for SPAdes without any denoising. Thus, depending on the inherent robustness of the assembler, the impact of denoising algorithms can be variable. Thus, we chose to work with SPAdes in this work. Most assemblers, including SPAdes, do not use quality scores. Hence, in this work, we do not attempt to denoise the quality scores.

Simulators

Although our main aim in the project was to denoise reads for assembly, we also experimented with simulated data to gain an understanding of the denoising process. For this purpose, we mostly used the pIRS simulator which uses a first order Markov model to generate quality scores, thus allowing non-independent noise within a read. It also models GC bias in the reads. We also ran a few experiments with ART, a popular simulator which generates independent noise with position-dependent distribution. ART was used by RECKONER for their evaluation [6].

We also used the align functionality provided by Karect to align the reads to the reference genome, treat the bases in the reference genome as the ground truth, and then compute the number of true positives etc. using this. This approach allows us to use real reads but also faces difficulties due to non-unique alignments and a sizeable fraction (~10%) of reads not getting aligned to the reference.

Real datasets

						Insert size	
Acc. No.	Species	Ploidy	Read length	Genome length	Coverage	Mean	Std. deviation
SRR543736	C Elegans	diploid	101	100M	58	173	13
SRR065390	C Elegans	diploid	100	100M	67	252	54
SRR823377	D melanogaster	diploid	100	116M	52	281	92
ERR204808	M acuminata	diploid	108	450M	15	337	51

Table 1: Real datasets.

Table 1 contains the real datasets used for the experiments. The insert size information was obtained using SPAdes. The datasets are selected from [1,4,6].

Approach 1: Using HARC for denoising

Github: https://github.com/shubhamchandak94/HARC/tree/denoising

We started this project thinking of improving compression while getting some denoising done on the side. The basic idea was to save some space by not storing the noise and noisepos for bases which appeared to be erroneous. However, we quickly realized that the saving on compression will be slight, especially for the order-preserving mode. Thus, we began to work with complete focus on the denoising problem.

In the encoder for HARC [7], any base that differs from the corresponding base in the majority-based reference contig is a candidate for potential denoising. Whether we denoise the base or not was decided based on a combination of counts and quality at the positions. We tried various ideas:

1. Counts only: denoise if counts of all bases except the reference base is less than *n**count of reference base. *n* is a parameter. This is a relative threshold because we are looking at the ratio of counts.
2. Quality only: denoise if quality of the base is below some threshold.
3. Counts and Quality: Denoise if both 1 and 2 are satisfied.
4. Adaptive: Denoise if #base_in_read/#base_in_reference < k*f(Q). Here k is a parameter and f is some decreasing function of Q.

All these ideas are per-base ideas, i.e., for the denoising decision we do not consider how well the rest of the read is aligned to the reference contig. These are very efficient but consider a case where reads from two repeating regions come together during the reordering. In this case, it would be more useful to denoise based on the reads closest to the current read and not based on all the overlapping reads. As an attempt to solve this problem, we tried some more schemes:

5. For a given read to be denoised, take the overlapping reads in the contig and apply weight = (1-sqrt(overlaperror/overlap/.25))*(1-P_e) while computing counts (this is based on Karect).
6. Instead of using all overlapping reads for denoising the current read, use the top *k* reads according to a score. We used scores which were linear combinations of the shift wrt the current read and the hamming distance to the current read.

Another observation from Karect was that they used absolute threshold on the count before denoising, rather than a relative threshold (as in point 1).

7. Use absolute threshold, i.e., denoise if count of base in read < *abs. thresh.*

Results and observations for approach 1

First, we list results and observations based on simulated reads, using undetected and new errors as the performance metric:

1. RECKONER beats the basic denoiser in point 1 on datasets simulated using ART. It appears that k-mer based methods are very suitable for reads with independent noise.
2. We generated three datasets using Pirs, each containing 8 million reads with length 100 each:
    a. From human chr22, no indels
    b. From human chr22, no indels, no GC bias
    c. From iid uniform genome of same length as human chr22, no indels, no GC bias

    We observed that on all the three datasets, HARC scheme 1 (counts only) was able to get close to RECKONER. For HARC, best parameters were: first stage threshold: 2, 4; second stage realignment threshold: 8; *n*: 0.1, 0.2. But it is interesting to see the performance of RECKONER on the 3 datasets (Table 2).

Dataset	Uncorrected Errors	New errors	Total
a	80K	196K	276K
b	63.5K	178K	241.5K
c	4.6K	0	4.6K

    Table 2: Performance of RECKONER on three simulated datasets

    For all the datasets, the number of errors introduced by the simulator was 2842K. Clearly, RECKONER corrects most of the errors (on ART simulated reads, RECKONER does even better). Removing the GC bias (which leads to non-uniform coverage) improves the error correction performance. But, sampling from an iid genome makes the denoising almost perfect. This shows that most of the difficulty in denoising arises from the repeats and non-iid nature of the genome.

3. Now we compare the various ideas listed above on a simulated dataset (25x *C. elegans*, pIRS simulator, 9450K errors). We list in table 3 a sample of the results with the corresponding parameters.

Denoiser	Parameters	Undetected errors	New errors	Total
Karect	Default	54	68	122
RECKONER	Default	74	117	191
Idea 1	0.1 relative thresh	320	380	700
Idea 2	30 quality thresh	470	180	650
Idea 4	f(Q) = P_e/(1-P_e), k=0.1	1682	717	2399
Idea 4	f(Q) = linear, k = 1 (0.01 at Q=40, 0.5 at Q=0)	150	358	508
Idea 4	f(Q) = exponential, k = 1 (0.05 at Q=40, 0.4 at Q=0)	155.4	331.3	486.7
Idea 5+7	Absolute thresh = 2.5	99.8	152.1	251.9
Idea 5+7	Absolute thresh = 3	92.7	204.4	297.1

    Table 3: Comparison of denoisers on simulated dataset. Numbers are in thousands. For all settings of HARC, stage 1 threshold is 4 and stage 2 realignment threshold is 8.

Some results not listed: Idea 3 does slightly better than ideas 1 and 2. Idea 6 in conjunction with 7 does slightly better than Idea 5+7, bringing the total errors to around 230K. Idea 3+7 is around 280K.

We see that taking absolute threshold drastically improves the performance, but even the best performance is worse than both RECKONER and Karect. Idea 4 with f(Q) = p_e/(1-p_e) should be the theoretically suggested use of quality values (using Bayes rule), but in practice, due to the small number of reads, using linear scaling in quality scores performs much better. One problem with the absolute threshold (idea 7) is its sensitivity. The correct threshold depends heavily on the coverage and even slight changes to the threshold can have great impact on the denoising performance.

Now, we look at some results on assembly. Table 4 shows the assembly performance for the 4 real datasets.

Dataset	Denoiser	N50	NG50	NA50	NGA50
SRR543736	None	7696	8559	5626	6347
SRR543736	Reckoner	7657	8522	5581	6296
SRR543736	Karect	7678	8512	5641	6319
SRR543736	Sickle	7944	8790	5797	6492
SRR543736	BayesHammer	**7992**	**8831**	**5832**	**6525**
SRR065390	None	25023	24408	21878	21255
SRR065390	Reckoner	24023	23139	20697	20108
SRR065390	Karect	**26001**	**25209**	**22618**	**21905**
SRR065390	Sickle	25042	24266	21577	20806
SRR065390	BayesHammer	25209	24476	21946	21337
SRR823377	None	65472	49455	50135	38680
SRR823377	Reckoner	52121	40465	41988	32449
SRR823377	Karect	71632	53688	**54045**	**41595**
SRR823377	Sickle	66592	50623	49920	38625
SRR823377	BayesHammer	**72001**	**54712**	51596	40232
ERR204808	None	8939	3726	8347	3433
ERR204808	Reckoner	8369	3397	7822	3148
ERR204808	Karect	**9456**	**3899**	**8982**	**3664**
ERR204808	Sickle	7931	3167	7608	2994
ERR204808	BayesHammer	8271	3214	7950	3077

Table 4: Performance of existing denoisers on assembly by SPAdes. Best results for each dataset are in boldface. The evaluation was done using Quast [13].

We see that RECKONER makes the assembly worse in almost all the cases, while Karect is typically the best denoiser. Using the trimmer Sickle usually makes little difference. For the low coverage ERR204808 dataset, trimming significantly worsens the assembly. Bayeshammer does better than RECKONER, and in some cases, beats Karect. Overall, we see that we can get an improvement of around 10% or lesser using these denoisers.

We also tried to run HARC based denoisers on these datasets and evaluate the assembly performance. On the SRR823377 dataset, we could get to N50 of 68152 using idea 3 (counts and quality threshold). This was the best score obtained after trying out all the ideas with a range of parameters. The later ideas like using absolute threshold, which were useful in reducing the number of errors, didn't help much in improving assembly. For several parameter settings, the assembly tends to become worse. On other datasets as well, we improved the assembly but didn't improve as much as Karect/Bayeshammer (whichever was better).

Before we get to approach 2, we mention a few results regarding the factors affecting assembly and simulators. Table 5 shows the assembly evaluation for a series of datasets, each containing length 100 paired end reads at a coverage of 15x from *M. acuminate* genome. The first dataset is real, while the others are simulated using pIRS

S. No.	Insert length		Comment	N50
	Mean	std		
1	337	51	Real dataset	8939
2	337	51	Noiseless	26448
3	337	51	Default pIRS profiles	24303
4	337	33.7	Default pIRS profiles	25780
5	180	18	Default pIRS profiles	16584
6	337	51	Noise profile from real	24633
7	337	51	Noise & GC bias profile from real	25310

Table 5: Assembly on real and simulated *M. acuminata* reads.

Note that even when we use both the noise and GC bias profiles from the real data and match the insert length characteristics, the N50 score for the simulated data is still around 3 times that for the real data. This seems to imply that there are certain characteristics of the real datasets that the simulators are unable to capture and that these differences make a huge impact in the assembly.

Comparing datasets 2 and 3, we see that there is around 10% scope of improvement in N50 by assembly, at least for the simulated datasets. After running Karect on 3, we get an N50 score of 24769. Thus, perhaps there is some scope of improvement.

We also see that the impact of insert length characteristics on assembly is much more significant than that of noise. Reducing the mean and/or standard deviation of the insert length seems to reduce the N50 score. For a detailed study of impact of various factors on assembly, please refer to [14], a technical note by Illumina.

Approach 2: Standalone denoiser

Github: https://github.com/shubhamchandak94/fastq_denoising

On visualizing some of the mistakes made by the approach 1 denoisers on simulated data, we observed that one major problem was that reads from the same region in the genome sometimes went into different contigs. This led to incorrect decisions due to the partial selection of reads used to denoise the current read.

To solve this, we decided to implement a standalone denoiser. Now, we start off by building hash tables of substrings of the read (as done in HARC). Then, we loop over the reads and for each read select the reads within some Hamming distance and up to some shift of the current read. Now, using this collection of reads, we attempt to denoise the current read. This algorithm is easily parallelizable, and the implementation supports multithreading. Note that this algorithm is quite similar to Karect with two differences: 1. Karect uses k-mers to search for overlapping reads while we use hashes of specific substrings of the reads (due to this, our memory consumption is much lower), 2. We tried various methods for the final step where we denoise the read using the overlapping reads.

1. We first tried Karect like technique for the final step, i.e., for each base in the current read, we accumulate counts from the overlapping reads, weighting them by their Hamming distance and quality scores (weighting factors same as in Karect). If the base with the largest count is different from the current base, we denoise if the count of the current base is greater than some absolute threshold. On the dataset in Table 3, this brought the undetected errors to 73K and the new errors to 9K (total 82K), thus beating Karect on this dataset.
2. On careful analysis of the mistakes made by 1, we observed that most reads have large number of exactly matching overlapping reads. However, we still make mistakes on those reads since we search for reads within some Hamming distance and hence pick up reads from other regions of the genome. Thus, we tried another idea:
    a. Sort the overlapping reads in increasing order according to (Hamming distance + 2)/Shift. The 2 in the numerator is just there to ensure that reads with 1 or 2 Hamming distance from current read but at 0 shift are prioritized over reads with 0 Hamming distance but large shifts.
    b. Now, for each base in the current read, start accumulating counts from the overlapping reads. Accumulate counts until (Count of some base + 0.25)/(Total count + 1) >= *threshold* for any one of the 4 bases (A, C, G, T). At this point, we decide that the current base should be the base satisfying the equation and stop accumulating counts for that base. Also, we apply this condition only after the Total count is greater than some parameter (around 3-4). In case the total count is smaller than this parameter even after all the overlapping reads are seen, we change the current base if there is overwhelming evidence against it (something like all other reads matching at that base).
    c. We observed that some of the reads had repeating k-mers within the read. In this situation, many of the overlapping reads matched the current read at multiple shifts. Earlier we considered such overlapping reads at only one shift, but that led to some new errors being introduced. Thus, we allowed such reads to occur multiple times in the list of overlapping reads.

In Table 6, we present some results for idea 2, run with the same set of parameters on several simulated datasets. We beat Karect on some datasets and RECKONER on all datasets. It is possible that using a better parameter selection, we might be able to beat Karect on all the datasets.

C. elegans 25x pirs					
	Total	FN	FP	Recall	Gain
Us	9454	10.5	6.4	**0.998889**	**0.998212**
Reckoner	9454	74.2	116.8	0.992151	0.979797
Karect	9454	53.7	67.53	0.99432	0.987177
C. elegans 50x pirs					
	Total	FN	FP	Recall	Gain
Us	18949	23.2	3.7	**0.998776**	**0.99858**
Reckoner	18949	118.6	233.1	0.993741	0.98144
Karect	18949	52.6	156.4	0.997224	0.98897
D. melanogaster 50x pirs					
	Total	FN	FP	Recall	Gain
Us	26832	233.7	833	0.99129	**0.960245**
Reckoner	26832	514.5	2018	0.980825	0.905616
Karect	26832	168.8	1303.3	**0.993709**	0.945136
C. elegans 67x ART	using per base noise statistics from SRR065390				
	Total	FN	FP	Recall	Gain
Us	281133	3117.4	360.7	0.988911	0.987628
Reckoner	281133	3847.6	1266.6	0.986314	0.981809
Karect	281133	1298.8	1552.8	**0.99538**	**0.989857**

Table 6: Some results on simulated data. We used a Hamming threshold of 16, relative threshold for the count ratio of 0.6. Total stands for total errors before denoising. All numbers are in 1000s.

We also tried to test approach 2 on assembly. However, the results were close to that of approach 1, still behind Karect. Table 7 lists some results for SRR065390 dataset. Observe that even though, idea 2 outperforms idea 1 on precision and recall, it falls behind on assembly. We also observed similar phenomenon with other datasets and even for real datasets where we measured precision and recall using the Karect align and eval tools.

Denoiser	N50	NG50	NA50	NGA50
None	25023	24408	21878	21255
Reckoner	24023	23139	20697	20108
Karect	**26001**	**25209**	**22618**	**21905**
Sickle	25042	24266	21577	20806
BayesHammer	25209	24476	21946	21337
Us – idea 1	25849	25069	22311	21632
Us – idea 2	25545	24784	22098	21565

Table 7: Some assembly results for SRR065390 *C. elegans* dataset.

Conclusions

In this project, we studied denoising of FASTQ files with an aim of improving assembly. This problem offered various challenges:

1. The assembly process is very complex and already includes several error correction mechanisms.
2. It is difficult to optimize for the metrics for evaluation of the assembly such as N50.
3. Often, improvement in precision and recall do not translate to an improvement in these metrics.
4. Assembly of simulated data seems much easier than real data, even when same noise and GC bias profiles are used. This makes drawing conclusions from simulated data very tricky.
5. Due to the lack of a useful upper bound, it is difficult to predict how much improvement is possible in assembly.

We suggested two approaches for denoising, one based on HARC and the other a standalone denoiser which uses hash tables to search for overlapping reads. With the second approach, we were able to beat the state-of-the-art denoiser Karect on simulated datasets. However, we are still behind Karect on assembly of real datasets.

References

1. Heydari, M., Miclotte, G., Demeester, P., Van de Peer, Y., & Fostier, J. (2017). Evaluation of the impact of Illumina error correction tools on de novo genome assembly. *BMC bioinformatics*, *18*(1), 374.
2. Bankevich, A., Nurk, S., Antipov, D., Gurevich, A. A., Dvorkin, M., Kulikov, A. S., ... & Pyshkin, A. V. (2012). SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing. *Journal of computational biology*, *19*(5), 455-477.
3. Nikolenko, S. I., Korobeynikov, A. I., & Alekseyev, M. A. (2013). BayesHammer: Bayesian clustering for error correction in single-cell sequencing. *BMC genomics*, *14*(1), S7.
4. Allam, A., Kalnis, P., & Solovyev, V. (2015). Karect: accurate correction of substitution, insertion and deletion errors for next-generation sequencing data. *Bioinformatics*, *31*(21), 3421-3428.
5. Długosz, M., & Deorowicz, S. (2017). RECKONER: read error corrector based on KMC. *Bioinformatics*, *33*(7), 1086-1089.
6. Długosz, M., Deorowicz, S., & Kokot, M. (2017, October). Improvements in DNA Reads Correction. In *International Conference on Man–Machine Interactions* (pp. 115-124). Springer, Cham.

7. Chandak, S., Tatwawadi, K., & Weissman, T. (2017). Compression of genomic sequencing reads via hash-based reordering: Algorithm and Analysis. *Bioinformatics*.
8. Hu, X., Yuan, J., Shi, Y., Lu, J., Liu, B., Li, Z., ... & Yue, Z. (2012). pIRS: Profile-based Illumina pair-end reads simulator. *Bioinformatics*, *28*(11), 1533-1535.
9. Huang, W., Li, L., Myers, J. R., & Marth, G. T. (2011). ART: a next-generation sequencing read simulator. *Bioinformatics*, *28*(4), 593-594.
10. Joshi NA, Fass JN. (2011). Sickle: A sliding-window, adaptive, quality-based trimming tool for FastQ files (Version 1.33) [Software]. Available at https://github.com/najoshi/sickle.
11. Sohn, J. I., & Nam, J. W. (2016). The present and future of de novo whole-genome assembly. *Briefings in bioinformatics*, bbw096.
12. Zerbino, D. R., & Birney, E. (2008). Velvet: algorithms for de novo short read assembly using de Bruijn graphs. *Genome research*, *18*(5), 821-829.
13. Gurevich, A., Saveliev, V., Vyahhi, N., & Tesler, G. (2013). QUAST: quality assessment tool for genome assemblies. *Bioinformatics*, *29*(8), 1072-1075.
14. De Novo Assmebly using Illumina Reads: https://www.illumina.com/Documents/products/technotes/technote_denovo_assembly_ecoli.pdf