NxRepair: Error Correction in De Novo Sequence Assembly Using Nextera Mate Pairs

Rebecca R. Murphy, Jared O'Connell, Anthony J. Cox and Ole Schulz-Trieglaff
Illumina Cambridge, Chesterford Research Park, Essex, CB10 1XL

November 14, 2014

Abstract

Supplementary Information for the paper NxRepair: Error Correction in De Novo Sequence Assembly Using Nextera Mate Pairs

1 Prior Work

Error correction in de novo assemblies is a well-studied problem. Recent work, such as the Assemblathon [1] and GAGE [2] collaborations compare the quality of assemblies prepared by various assemblers. A Bayesian method of assembly quality evaluation also exists [3]. Several recent papers have developed error identification and correction methods. The A5 Assembly Pipeline [4] includes an error detection and rescaffolding step and two new tools, REAPR [5] and ALE [6] use read pair data to identify misassemblies. A similar tool is currently under development at the Broad Institute [7]. NxRepair is complementary to these tools, as it specifically uses Nextera mate-pair information to find the largest and most serious misassemblies: REAPR and A5-MiSeq do not use mate-pair information and ALE is unfortunately no longer in active development.

2 Supplementary Methods

2.1 Data

Nine bacterial genomes were prepared according to the Nextera Mate Pair protocol and sequenced in a single MiSeq run using 2×151 bp reads. The genomes sequenced are shown in Table 1. Reads were trimmed using the MiSeq inbuilt trimmer. The untrimmed reads are available from BaseSpace via `https://basespace.illumina.com/s/TXv32Ve6wTl9` (free registration required). Note that only these Nextera mate pair libraries were used. No additional single end or paired end libraries were required.

2.2 Workflow Pipeline

De novo assemblies were prepared using the SPAdes Assembler, version 3.1.1 [8]:

```
spades.py -k 21,33,55,77 -t 4 --careful --hqmp1-12 bacteria.fastq.gz --hqmp1-fr
-o assembly
```

The initial assembly quality was evaluated using QUAST [9] to aligning the de novo assembly to a reference genome:

```
python quast.py -o results_sample -t 16 -R ref/referenece.fna sample_new.fasta
```

Following assembly of the remaining seven genomes, the mate-pair reads were aligned back to the de novo assembly using BWA-MEM [10]. A sorted bam file of the resulting alignment was then prepared using SAMtools [11]:

```
bwa index sample/scaffolds.fasta
```

```
bwa mem sample/scaffolds.fasta -p bacteria.fastq.gz | samtools view -bS - | samtools
sort - sample
```

```
samtools index sample.bam
```

We identified misassemblies using NxRepair as follows:

```
python nxrepair.py sample.bam sample/scaffolds.fasta sample_scores.csv sample_new.fasta
-img_name sample_new
```

The default parameters used and their meanings are shown in Table 2. These have been optimised for Illumina Nextera mate-pair libraries with a mean insert size of approximately 3 Kb. For mate-pair libraries with a much larger (smaller) insert size, the maxinsert and trim parameters may need to be increased (decreased).

Finally we used QUAST [9] to evaluate the assembly quality following NxRepair by aligning the de novo assembly to a reference genome as described above.

3 Implementation

3.1 Global Assembly Parameters

NxRepair identifies misassemblies based on the insert size distribution of mate pairs aligned to the de novo assembly. The first step is calculation of the global mate pair insert size distribution. For calculation of population statistics, mate pairs that align to different contigs from each other are excluded, as are mate pairs with an incorrect strand or pairing orientation, pairs whose mapping quality falls below a user specified threshold, and pairs whose insert size exceed 30 Kb (approximately 10 times the mean insert size). The global mean $\hat{\mu}$ and median absolute deviation MAD were calculated across all contigs in the assembly as:

$$\hat{\mu} = \frac{\sum_{i=1}^{N} Y_i}{N} \qquad \text{MAD} = \text{median}_i(|Y_i - \text{median}_j(Y_j)|) \tag{1}$$

where Y_l is the insert size of the ith of N reads with correct pairing behaviour. These were then used as the parameters of the null distribution, as described in the main paper. The median absolute deviation was used in place of the standard deviation, as it is more robust to outlier insert sizes.

3.2 Interval Tree Construction

To facilitate rapid lookup of mate pair properties, we construct an interval tree [12] for each contig in the de novo assembly. An interval tree is a datastructure that facilitates $O(\log n + m)$ lookup of intervals that span a given point or interval, for n total entries and m spanning entries. The interval tree contains the start and end positions of each mate pair aligned to that contig, as well as an flag variable indicating whether that mate pair had correct strand and pairing orientation. Mate pairs where the two pairs aligned to different contigs were excluded, This allows NxRepair to rapidy query positions across a contig to discover the insert size distribution at the queried position.

3.3 Assembly Support and Z-Score Calculation

To calculate the degree of support for the assembly at each site across a contig, NxRepair retrieves all mate pairs spanning a window of size 'window' at position i on the contig. As described in the main paper, we define a latent indicator variable $X_i \in \{0, 1\}$ for each pair of reads, l, which takes the value 1 if the insert size came from our null distribution, and 0 otherwise. Within each window queried, the probability that each retrieved read, r_i is drawn from the null distribution is given by:

$$\begin{aligned} P(X_i = x|Y_i) &= \frac{P(X_i = x)(Y_i|X_i = x)}{\sum_{k=0}^{1} P(X_i = k)(Y_i|X_i = k)} && (2) \\ &= \frac{\pi_x(Y_i|X_i = x)}{\sum_{k=0}^{1} \pi_k(Y_i|X_i = k)} && (3) \end{aligned}$$

where π_x is the user defined prior probability of class x and $\pi_1 + \pi_0 = 1$. The default value of π_0 is 0.01 (see table 2), meaning that in the absence of any insert size information, 99 % of read pairs are from the null distribution.

Within each window, the total support for a correct assembly at position l can be calculated as:

$$D_l = \sum_{i=1}^{N} P(Z_i = 1|Y_i) \cdot C_i \tag{4}$$

where C_i is an indicator variable, reporting pairing orientation:

$$C_i = \begin{cases} 1, & \text{if mate pairs have correct orientation and strand alignment} \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

Within each contig, the contig assembly support mean $\hat{\mu}_D$ and variance s_D are calculated from all reads aligning to the contig,

$$\hat{\mu}_D = \frac{\sum_{l=1}^{N} D_l}{N} \qquad s_D = \frac{\sum_{l=1}^{N} \sqrt{(D_l - \hat{\mu})^2}}{N} \tag{6}$$

Using these values, the Z-score z_l within each queried interval is calculated as:

$$z_l = \frac{D_l - \hat{\mu}_D}{s_D} \tag{7}$$

A misassembly is identified if $z_l < T$ for a user-defined threshold T (default value -4). This threshold describes the number of standard deviations below the mean assembly support that is required to identify an anomaly. The default value of -4 will flag only positions whose assembly support is less than four standard deviations below the mean level of support. As the parameters of the distribution are derived from the properties of the assembly, this is robust to variation in coverage, contig size and other assembly properties.

3.4 Misassembly Location and Contig Breaking

To improve the quality of the de novo assembly, a contig is broken into two separate pieces at the site of a misassembly and the broken ends of the two new contigs trimmed by a user defined length (default 4 Kb) to remove the misassembled region. To prevent excessive clipping, misassemblies separated by less than the trimming distance are grouped together, the contig is broken at the start and end of the misassembled region and the misassembled section is discarded. Low-scoring regions within the trimming distance of the ends of contigs are not considered misassemblies, as the high proportion of mate pairs aligning here whose mate maps to a different contig reduces the number of pairs under consideration and hence lowers the observed Z-score.

3.5 Evaluation

To evaluate the performance of NxRepair, we prepared Receiver Operating Curves (ROCs) for each bacterial genome, evaluated at a range of different thresholds. As misassemblies

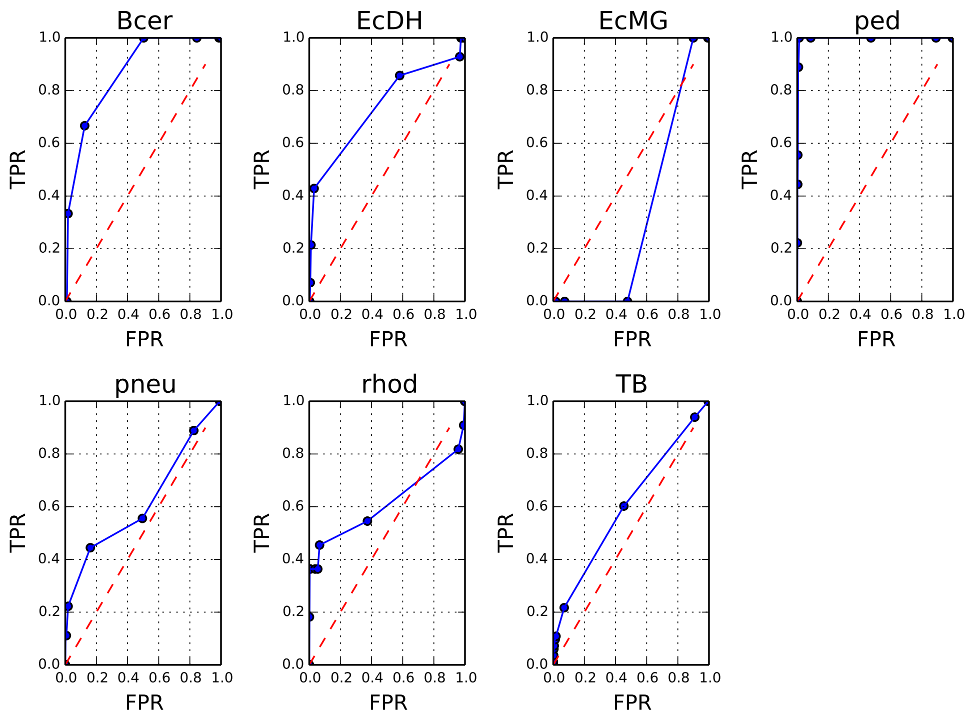

Figure 1: ROC plots for each of the seven genomes evaluated here.

are identified as point errors, but our NxRepair method identifies the region spanned by a misassembly, we needed a method to correctly compare the sites of true misassemblies with those identified by NxRepair. To make this comparison, we divided each contig of the assembly into short stretches of 1 Kb length. We then prepared an array, A_{Nx} of size $\frac{L}{1000}$ for contig length L, corresponding to misassemblies identified by NxRepair. A_{Nx} was filled as follows:

$$A_{Nx} = \begin{cases} 1, & \text{if NxRepair identified a misassembly in stretch } i \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

To prepare the ROCs each position i in A_{Nx} was labeled as true positive (TP) if $A_{Nx}[i] = 1$ and a true misassembly fell within it, true negative (TN) if $A_{Nx}[i] = 0$ and no true misassembly occurred within the interval, false positive (FP) if $A_{Nx}[i] = 1$ but no true misassembly had occurred, or false negative (FN) if $A_{Nx}[i] = 0$ but the interval contained a true misassembly. The true positive rate (TPR) false positive rate (FPR) were then calculated as follows:

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} \qquad \text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{9}$$

The ROC plots are shown in Supp. Figure 1.

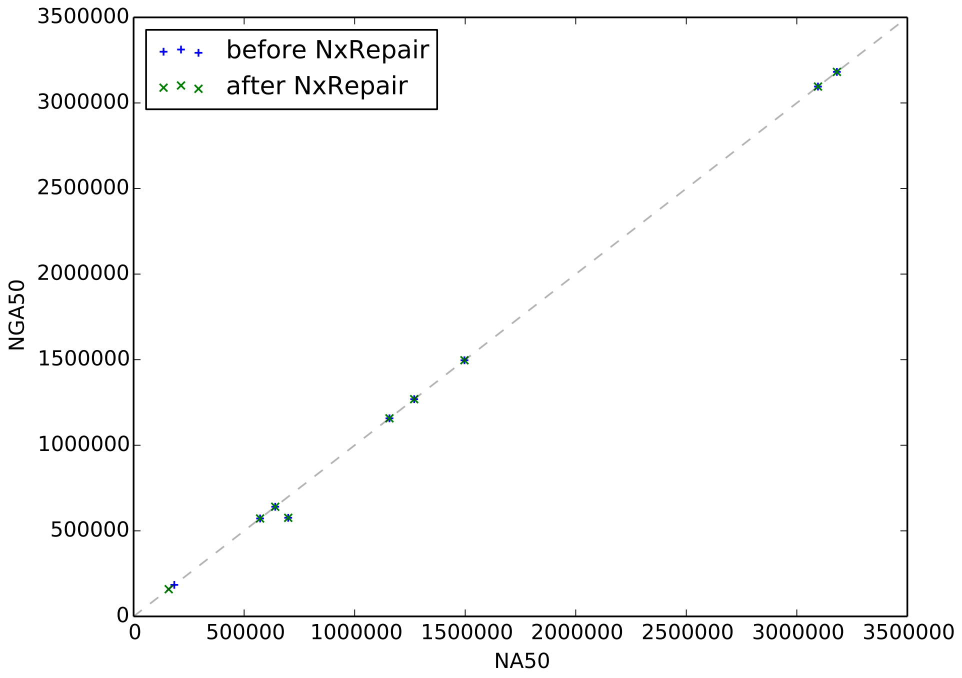

Figure 2: Plot of NGA50 vs NA50 for each genome before and after NxRepair correction.

Finally, we plotted the NGA50 value as calculated by QUAST against the NA50, before and after NxRepair correction, to demonstrate that we have not reduced the assembly quality. This is shown in Supp. Figure 2.

4 Performance

We evaluated the runtime and peak memory usage of NxRepair on each of the nine genomes analysed. The results are shown in table 3. Performance analysis was performed on a single core with GB RAM available. Runtime analysis was performed using the python cProfile module. The memoryprofiler python module was used to analyse memory usage. The most memory and compuationally intensive part of the NxRepair analysis is construction of the interval trees. The size of each interval tree constructed is dependent on the contig size. Consequently, we expect both runtime and memory usage to scale with the size of the largest contig.

Abbreviation:	Bcer
Bacteria:	*Bacillus cereus ATCC 10987*
Accession ID:	NC_003909, NC_005707
NCBI FTP:	`ftp.ncbi.nih.gov/genomes/Bacteria/Bacillus_cereus_ATCC_10987_uid57673/`
Abbreviation:	EcDH
Bacteria:	*Escherichia coli str. K-12 substr. DH10B*
Accession ID:	NC_010473
NCBI FTP:	`ftp.ncbi.nih.gov/genomes/Bacteria/Escherichia_coli_K_12_substr__DH10B_uid58979/`
Abbreviation:	EcMG
Bacteria:	*Escherichia coli str. K-12 substr. MG1655*
Accession ID:	NC_000913
NCBI FTP:	`ftp.ncbi.nih.gov/genomes/Bacteria/Escherichia_coli_K_12_substr__MG1655_uid57779/`
Abbreviation:	list
Bacteria:	*Listeria monocytogenes*
Accession ID:	NC_003210
NCBI FTP:	`ftp.ncbi.nih.gov/genomes/Bacteria/Listeria_monocytogenes_EGD_e_uid61583/`
Abbreviation:	meio
Bacteria:	*Meiothermus ruber DSM 1279*
Accession ID:	NC_013946
NCBI FTP:	`ftp.ncbi.nih.gov/genomes/Bacteria/Meiothermus_ruber_DSM_1279_uid46661/`
Abbreviation:	ped
Bacteria:	*Pedobacter heparinus DSM 2366*
Accession ID:	NC_013061
NCBI FTP:	`ftp.ncbi.nih.gov/genomes/Bacteria/Pedobacter_heparinus_DSM_2366_uid59111/`
Abbreviation:	pneu
Bacteria:	*Klebsiella pneumoniae subsp. pneumoniae MGH 78578*
Accession ID:	NC_009648, NC_009649, NC_009650, NC_009651, NC_009652, NC_009653
NCBI FTP:	`ftp.ncbi.nih.gov/genomes/Bacteria/Klebsiella_pneumoniae_MGH_78578_uid57619/`
Abbreviation:	rhod
Bacteria:	*Rhodobacter sphaeroides 2.4.1*
Accession ID:	NC_007488, NC_007489, NC_007490, NC_007493, NC_007494, NC_009007, NC_009008
NCBI FTP:	`ftp.ncbi.nih.gov/genomes/Bacteria/Rhodobacter_sphaeroides_2_4_1_uid57653/`
Abbreviation:	TB
Bacteria:	*Mycobacterium tuberculosis H37Ra*
Accession ID:	NC_009525
NCBI FTP:	`ftp.ncbi.nih.gov/genomes/Bacteria/Mycobacterium_tuberculosis_H37Ra_uid58853/`

Table 1: Summary of bacteria analysed and the relevant NCBI information on their reference genomes. There were two repeats of each strain. All 18 samples were prepared with the Nextera Mate Pair protocol and sequenced in a single MiSeq run using 2×151 bp reads. The untrimmed reads we used as input to NxTrim (3.9Gbp in all) are available from BaseSpace via `https://basespace.illumina.com/s/TXv32Ve6wTl9` (free registration required).

Parameter	Default Value	Meaning
imgname	None	Prefix under which to save plots.
maxinsert	30000	Maximum insert size, below which a read pair is included in calculating population statistics.
minmapq	40	Minimum MapQ value, above which a read pair is included in calculating population statistics.
minsize	10000	Minimum contig size to analyse.
prior	0.01	Prior probablility that the insert size is anomalous.
stepsize	1000	Step-size in bases to traverse contigs.
trim	4000	Number of bases to trim from each side of an identified misassembly.
T	-4.0	Threshold in Z score (number of standard deviations from the mean) below which a misassembly is called.
window	200	Window size across which bridging mate pairs are evaluated.

Table 2: NxRepair Parameters

Bacterium	Total Time (s)	Memory Usage (MiB)
Bcer	78	271
EcDH	123	444
EcMG	70	260
list	97	383
meio	259	565
ped	123	417
pneu	59	227
rhod	190	463
TB	155	411

Table 3: NxRepair performance analysis.

5 Availibility and Dependencies

NxRepair is available for free anonymous download from the Python Package Index (PyPI) here: `:https://pypi.python.org/pypi/nxrepair`. The source code, written in python is hosted on GitHub: `https://github.com/rebeccaroisin/nxrepair`. A full tutorial and API can be found on ReadTheDocs: `http://nxrepair.readthedocs.org/en/latest/`.

NxRepair makes use of several further open source libraries, specifically:

Numpy [13] (`http://www.numpy.org/`)

Scipy [14] (`http://www.scipy.org/`)

Matplotlib [15] (`http://matplotlib.org/`)

Pysam (`https://pypi.python.org/pypi/pysam`), the python wrapper for Samtools

Samtools [11] (`http://samtools.sourceforge.net/`)

We installed the numpy, scipy and matplotlib libraries via Anaconda (`https://store.continuum.io/cshop/anaconda/`).

We have used the Interval Tree implementation from the bx-python library.

References

[1] K.R. Bradnam, J. N. Fass, A. Alexandrov, P. Baranay, M. Bechner, I. Birol, S. Boisvert, J. A. Chapman, G. Chapuis, R. Chikhi, H. Chitsaz, W. C. Chou, J. Corbeil, C. Del Fabbro, T. R. Docking, R. Durbin, Earl D., S. Emrich, P. Fedotov, N. A. Fonseca, G. Ganapathy, R. A. Gibbs, S. Gnerre, E. Godzaridis, S. Goldstein, M. Haimel, G. Hall, D. Haussler, J. B. Hiatt, I. Y. Ho, J. Howard, M. Hunt, S. D.. Jackman, D. B. Jaffe, E. D. Jarvis, H. Jiang, S. Kazakov, P. J. Kersey, J.O. Kitzman, J.R. Knight, S. Koren, T. W. Lam, D. Lavenier, F. Laviolette, Y. Li, Z. Li, B. Liu, Y. Liu, R. Luo, I. Maccallum, M.D. Macmanes, N. Maillet, S. Melnikov, D. Naquin, Z. Ning, T. D. Otto, B. Paten, O.S. Paulo, A. M. Phillippy, F. Pina-Martins, M. Place, D. Przybylski, X. Qin, C. Qu, F. J. Ribeiro, S. Richards, D. S. Rokhsar, J. G. Ruby, S. Scalabrin, M. C. Schatz, D. C. Schwartz, A. Sergushichev, T. Sharpe, T. I. Shaw, J. Shendure, Y. Shi, J. T. Simpson, H. Song, F. Tsarev, F. Vezzi, R. Vicedomini, B. M. Vieira, J. Wang, K. C. Worley, S. Yin, S. M. Yiu, J. Yuan, G. Zhang, H. Zhang, S. Zhou, and I. F. Korf. Assemblathon 2: evaluating de novo methods of genome assembly in three vertebrate species. *Gigascience*, 2(1):10, 2013.

[2] S. L. Salzberg, A. M. Phillippy, A. Zimin, D. Puiu, T. Magoc, S. Koren, T. J. Treangen, M. C. Schatz, A. L. Delcher, M. Roberts, G. Marais, M. Pop, and J. A. Yorke. Gage: A critical evaluation of genome assemblies and assembly algorithms. *Genome Res.*, 22(3):557–567, 2012.

[3] M. Ghodsi, C. M. Hill, I. Astrovskaya, H. Lin, and D. D. Sommer. Se novo likelihood-based measures for comparing genome assemblies. *BMC Research Notes*, 6:334, 2013.

[4] D. Coil, G. Jospin, and A. E. Darling. A5-miseq: an updated pipeline to assemble microbial genomes from illumina miseq data. 2014.

[5] M. Hunt, T. Kikuchi, M. Sanders, C. Newbold, M. Berriman, and T. D. Otto. Reapr: a universal tool for genome assembly evaluation. *Genome Biol.*, 14:R47, 2013.

[6] S. C. Clark, R. Egan, P. I. Frazier, and Z. Wang. Ale: a generic assembly likelihood evaluation framework for asessing the accuracy of genome and metagenome assemblies. *Bioinformatics*, 29(4):435–443, 2013.

[7] B. Walker. Pilon. `https://github.com/broadinstitute/pilon/releases`, 2014.

[8] A. Bankevich, S. Nurk, D. Antipov, A. A. Gurevich, M. Dvorkin, A. S. Kulikov, V. M. Lesin, S. I. Nikolenko, S. Pham, A. D. Prjibelski, A. V. Pyshkin, A. V. Sirotkin, N. Vyahhi, G. Tesler, M. A. Alekseyev, and P. A. Pevzner. Spades: A new genome assembly algorithm and its applications to single-cell sequencing. *J. Comp. Biol.*, 19(5):455–477, 2012.

[9] A. Gurevich, V. Saveliev, N. Vyahhi, and G. Tesler. Quast: quality assessment tool for genome assemblies. *Bioinformatics*, 29(8):1072–1075, 2013.

[10] H. Li. Aligning sequence reads, clone sequences and assembly contigs with bwa-mem. 2013.

[11] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, R. Durbin, and 1000 Genome Project Data Processing Subgroup. The sequence alignment/map format and samtools. *Bioinformatics*, 25(16):2078–2079, 2009.

[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press and McGraw-Hill, 2009.

[13] S. van der Walt, S. C. Colbert, and G. Varoquaux. The numpy array: A structure for efficient numerical computation. *Computing in Science and Engineering*, 13:22–30, 2011.

[14] K. J. Millman and M. Aivazis. Python for scientists and engineers. *Computing in Science and Engineering*, 13:9–12, 2011.

[15] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science and Engineering*, 9(3):90–95, 2007.