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

Rebecca R. Murphy[1], Jared O'Connell[2], Anthony J. Cox[3], and Ole Schulz-Trieglaff[4]

[1]**Illumina Cambridge, Chesterford Research Park, Essex, CB10 1XL**
[2]**Illumina Cambridge, Chesterford Research Park, Essex, CB10 1XL**
[3]**Illumina Cambridge, Chesterford Research Park, Essex, CB10 1XL**
[4]**Illumina Cambridge, Chesterford Research Park, Essex, CB10 1XL**

ABSTRACT

Scaffolding errors and incorrect traversals of the de Bruijn graph during de novo assembly can result in large scale misassemblies in draft genomes. Nextera mate pair sequencing data provide additional information to resolve assembly ambiguities during scaffolding. Here, we introduce NxRepair, an open source toolkit for error correction in de novo assemblies that uses Nextera mate pair libraries to identify and correct large-scale errors. We show that NxRepair can identify and correct large scaffolding errors, without use of a reference sequence, resulting in quantitative improvements in the assembly quality. NxRepair can be downloaded from GitHub; a tutorial and user documentation are also available.

Keywords: de novo assembly, bioinformatics, error correction, bacteria

INTRODUCTION

De Bruijn Graph construction and traversal is a popular method for de novo genome assembly (Compeau et al., 2011). However, traversal of repeat regions, which tangle the de Bruijn Graph, remains challenging. Read pairs with a large insert size, such as the Illumina Nextera mate pairs can provide additional information for repeat disambiguation. Many assemblers incorporate mate pair insert size information into the assembly and scaffolding process (Bankevich et al., 2012; Zerbino and Birney, 2008), but large scale scaffolding errors can still occur (Fig. 1 (A)).

Error correction in de novo assemblies is a well-studied problem. Recent work, such as the Assemblathon (Bradnam et al., 2013) and GAGE (Salzberg et al., 2012) collaborations, compare the quality of assemblies prepared by various assemblers. A Bayesian method of assembly quality evaluation also exists (Ghodsi et al., 2013). Several recent papers have developed error identification and correction methods. The A5 Assembly Pipeline (Coil et al., 2014) includes an error detection and rescaffolding step and two new tools, REAPR (Hunt et al., 2013) and ALE (Clark et al., 2013) use read pair data to identify misassemblies. A similar tool is currently under development at the Broad Institute (Walker, 2014). However with the exception of ALE, which is no longer actively maintained, these tools are not optimised to use mate pair information.

Here we introduce NxRepair, an assembly error detection tool that can identify the most serious misassemblies, without using a reference sequence, by examining the distribution of Nextera mate pair insert sizes. NxRepair specifically targets the most serious misassemblies by identifying regions with a high number of anomalous insert sizes, breaking the scaffold and optionally trimming out the misassembled region. NxRepair is complementary to existing tools, as it specifically uses Nextera mate pair information to find the largest and most serious misassemblies.

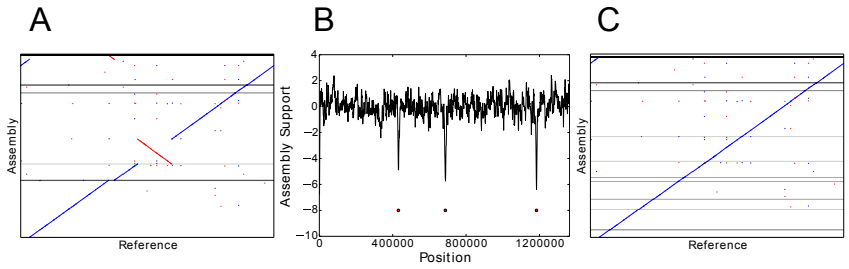

Figure 1. Using NxRepair to remove large misassemblies. (A) A de novo assembly of the Mycobacterium tuberculosis genome contains several large misassemblies. (B) Low support for the assembly is identified in two regions using NxRepair. (C) Breaking the contigs at the identified positions resolves the most significant misassemblies. In (A) and (C), horizontal lines demarcate contig boundaries.

IMPLEMENTATION

Statistical Analysis of Mate Pair Insert Sizes

Nextera mate pair libraries are prepared to have a certain insert size, typically between 1 and 10 kb. When the mate pairs used to prepare an assembly are aligned back to the assembly, large misassemblies result in unusual insert sizes and read orientations. We model this using a two-component mixture distribution. The first component of this mixture is the insert size distribution of correctly aligned mate pairs. We model the distribution of insert sizes, Y, as a normal distribution with mean $\hat{\mu}$ and standard deviation $\hat{\sigma}$: $Y \sim N(\hat{\mu}, \hat{\sigma}^2)$. We estimate $\hat{\mu}$ and $\hat{\sigma}$ for the entire genome by aligning reads back to the assembly and using robust estimators (see below). The second component, defined as a uniform distribution across the contig size $U(0, L)$ for a contig of length L, captures anomalous insert sizes.

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. We define a latent indicator variable $X_l \in \{0, 1\}$ for each pair of reads, l, which takes the value 1 if the insert size came from the null distribution, and 0 otherwise. Within each window queried, the probability that each retrieved read, r_l is drawn from the null distribution is given by:

$$P(X_l = x|Y_l) = \frac{P(X_l = x)(Y_l|X_l = x)}{\sum_{k=0}^{1} P(X_l = k)(Y_l|X_l = k)} \quad (1)$$

$$= \frac{\pi_x(Y_l|X_l = x)}{\sum_{k=0}^{1} \pi_k(Y_l|X_l = k)} \quad (2)$$

where Y_l is the insert size of read pair l, 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 3), meaning that in the absence of any insert size information, 99 % of read pairs are expected to arise from the null distribution.

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

$$D_i = \sum_{l=1}^{N} P(X_l = 1|Y_l) \cdot C_l \quad (3)$$

where the summation is over all read pairs aligning across position i and C_l is an indicator variable, reporting pairing orientation:

$$C_l = \begin{cases} 1, & \text{if mate pairs have correct orientation and strand alignment} \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

Within each contig, the contig assembly support mean μ_D and variance s_D are calculated from all reads aligning to the contig,

$$\hat{\mu}_D = \frac{\sum_{l=1}^{N} D_l}{N} \qquad s_D = \frac{\sum_{l=1}^{N} \sqrt{(D_l - \hat{\mu})^2}}{N} \tag{5}$$

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

$$z_l = \frac{D_l - \hat{\mu}_D}{s_D} \tag{6}$$

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.

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).

Global Assembly Parameters

NxRepair identifies misassemblies by identifying regions where the mate pair insert size distribution differs significantly from the insert size distribution across the entirety of the de novo assembly. Consequently,

it is necessary to have a robust estimate of the global mate pair insert size distribution. For calculation of population statistics, mate pairs that align to different contigs 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 are calculated across all contigs in the assembly as:

$$\hat{\mu} = \frac{\sum_{l=1}^{N} Y_l}{N} \qquad \text{MAD} = \text{median}(|Y_l - \text{median}(Y_l)|) \tag{7}$$

where Y_l is the insert size of the lth of N reads with correct pairing behaviour. The standard deviation was then calculated from the MAD, using:

$$\hat{\sigma} = K \cdot \text{MAD} \tag{8}$$

for $K = 1.4826$

These were then used as the parameters of the null distribution, as described in the main paper.

Interval Tree Construction

To facilitate rapid lookup of mate pair properties, we construct an interval tree (Cormen et al., 2009) 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 reads align to different contigs were excluded. This allows NxRepair to rapidly query positions across a contig to discover the insert size distribution at the queried position.

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.

Availability 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 (van der Walt et al., 2011) (`http://www.numpy.org/`)

Scipy (Millman and Aivazis, 2011) (`http://www.scipy.org/`)

Matplotlib (Hunter, 2007) (`http://matplotlib.org/`)

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

Samtools (Li et al., 2009) (`http://samtools.sourceforge.net/`)

	Before NxRepair		After NxRepair	
Genome	No.	NGA50	No.	NGA50
Bcer	3	1157404	3	1157404
EcDH	8	576143	8	576143
EcMG	2	640732	2	640732
List	0	1496615	0	1496615
Meio	0	3095733	0	3095733
ped	6	1269259	0	1269259
pneu	7	577220	6	577220
Rhod	9	3181390	9	3181390
TB	70	184170	66	158885

Table 2. Number of large misassemblies and NGA50 as reported by QUAST before and after NxRepair correction.

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 (`https://bitbucket.org/james_taylor/bx-python/wiki/Home`).

MATERIALS AND METHODS

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.

Performance Optimisation

ROC Plots

To optimimse the threshold in Z below which to identify a misassembled region, we prepared ROC plots, varying the threshold value, T, in steps of 1 between -10 and 0.

As misassemblies are identified as point errors, but NxRepair 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:

$$A_{Nx} = \begin{cases} 1, & \text{if NxRepair identified a misassembly in stretch } i \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

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

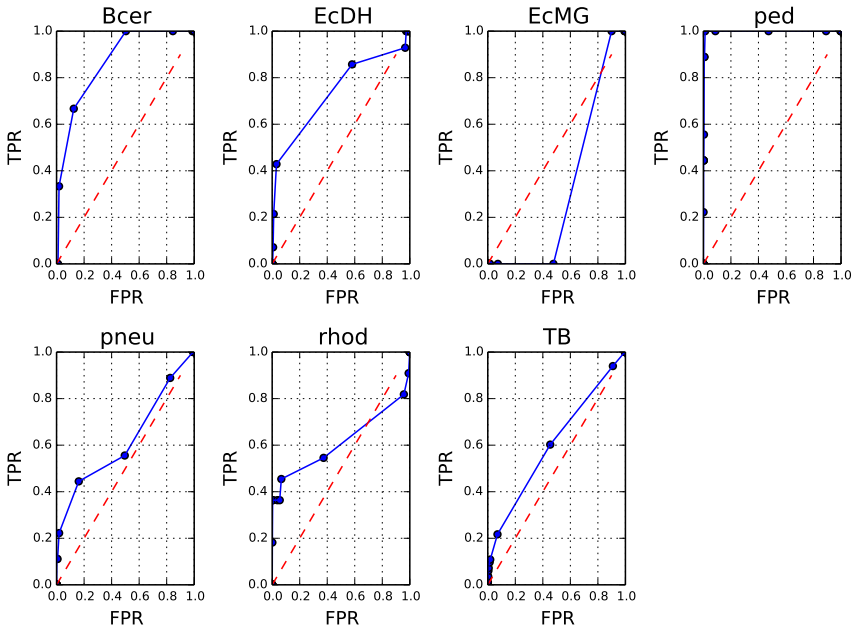

Figure 2. ROC plots for the seven genomes containing misassemblies.

$A_{Nx}[i] = 0$ but the interval contained a true misassembly. The true positive rate (TPR) and false positive rate (FPR) were then calculated as follows:

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} \qquad \text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{10}$$

Based on the resultant ROC plots, shown in Figure 2, a threshold in Z of -4 was found to optimise the true positive: false positive ratio. This threshld was used for all subsequent analyses.

Profiling

Performance analysis was performed on a single core with 8 GB RAM available. Runtime analysis was performed using the python cProfile module. The memoryprofiler python module was used to analyse memory usage.

Workflow Pipeline

De novo assemblies were prepared using the SPAdes Assembler, version 3.1.1 (Bankevich et al., 2012):

```
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 (Gurevich et al., 2013) to align the de novo assembly to a reference genome:

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

Following assembly, the mate pair reads were aligned back to the de novo assembly using BWA-MEM (Li, 2013). A sorted BAM file of the resulting alignment was then prepared using SAMtools (Li et al., 2009):

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

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 3. NxRepair Parameters

```
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 3. 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 (Gurevich et al., 2013) to evaluate the assembly quality following NxRepair by aligning the de novo assembly to a reference genome as described above.

RESULTS AND DISCUSSION

We used NxRepair to correct de novo assemblies from nine bacterial genomes. The genomes used are described above. Mate pair reads were trimmed, assembled using the SPAdes assembler (version 3.1.1) (Bankevich et al., 2012) and then aligned back to the assembled scaffold using BWA-MEM (Li, 2013). We used QUAST (Gurevich et al., 2013) to evaluate the assembly quality before and after NxRepair correction by aligning to an appropriate reference genome. For all NxRepair analyses, the default parameters, shown in Table 3 were used. Fig. 1 (A) shows a misassembled genome that contained several scaffolding errors identified by NxRepair (Fig. 1 (B)). Following NxRepair correction, the most significant structural misassemblies were resolved (Fig. 1 (C)). The improvement following NxRepair correction is shown for all nine genomes in Table 2. For two assemblies, errors were removed without reducing NGA50; for one genome, errors were removed but NGA50 was slightly reduced; for five genomes, two of which contained no large errors, no errors were found and the assembly was unchanged.

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 Figure 3. We note that in the case of the TB genome, the NGA50 was slightly reduced by NxRepair correction. Manual inspection of the correction sites revealed that one of the misassemblies reported by nxrepair was a join between two contigs which consisted of a gap of over 2 Kb bridged by very few mate pairs. This join is reported as correct by Quast, but has only little support from the read data. We fail to

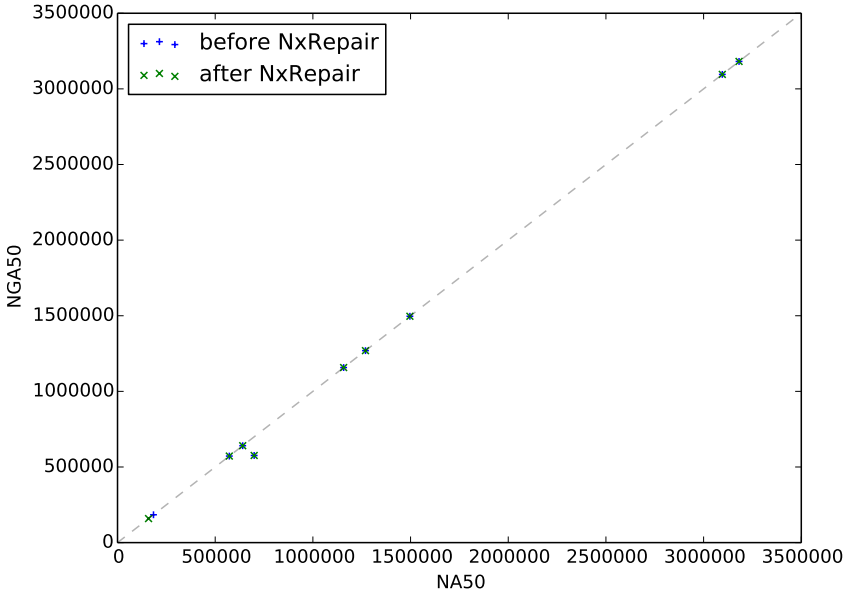

Figure 3. Plot of NGA50 vs NA50 for each genome before and after NxRepair correction.

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 4. NxRepair performance analysis.

correct some misassemblies because they do not exhibit a signal given the wide insert size distribution of the Nextera mate pairs.

Performance

We evaluated the runtime and peak memory usage of NxRepair on each of the nine genomes analysed. The results are shown in table 4. The most memory and computationally intensive part of the NxRepair analysis is construction of the interval trees. The size of each interval tree is dependent on the contig size. Consequently, we expect both runtime and memory usage to scale with the size of the largest contig.

CONCLUSIONS

NxRepair is a simple error correction module that can be used to identify and remove large scale errors from de novo assemblies using Nextera mate pair reads. We evaluated NxRepair using nine bacterial genomes, showing that of the seven genomes containing misassemblies, six could be improved by NxRepair correction. NxRepair is freely available online and can be run with a single call from the command line, making it an attractive option for improving assembly quality.

ACKNOWLEDGMENTS

REFERENCES

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

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

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

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

Compeau, P. E. C., Pevzner, P. A., and Tesler, G. (2011). How to apply de bruijn graphs to genome assembly. *Nature Biotechnol.*, 29:987–991.

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

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

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

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

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

Li, H. (2013). Aligning sequence reads, clone sequences and assembly contigs with bwa-mem.

Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., Durbin, R., and Subgroup, . G. P. D. P. (2009). The sequence alignment/map format and samtools. *Bioinformatics*, 25(16):2078–2079.

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

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

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

Walker, B. (2014). Pilon. `https://github.com/broadinstitute/pilon/releases`.

Zerbino, D. R. and Birney, E. (2008). Velvet: algorithms for de novo short read assembly using de bruijn graphs. *Genome Res.*, 18(5):821–829.