Suffix-Tree Based Error Correction of NGS Reads Using Multiple Manifestations of an Error

Daniel M. Savel
Electrical Engineering and Computer Science
Case Western Reserve University
Cleveland, OH
dan.savel@case.edu

Thomas LaFramboise
(1) Genetics and Genome Sciences
(2) Center for Proteomics and Bioinformatics
Case Western Reserve University
Cleveland, OH
thomas.laframboise@case.edu

Ananth Grama
Dept. of Computer Science
Purdue University
West Lafayette, IN
ayg@cs.purdue.edu

Mehmet Koyutürk
(1) Electrical Engineering and Computer Science
(2) Center for Proteomics and Bioinformatics
Case Western Reserve University
Cleveland, OH
mehmet.koyuturk@case.edu

ABSTRACT

Next Generation Sequencing (NGS) technologies produce large quantities of short length reads with higher error rates. Erroneous reads that cannot be aligned, are either ignored during *de-novo* sequencing, or must be suitably corrected. Such reads pose problems for mapping as well, since it is difficult to distinguish errors from true variants. Methods for detection and correction of errors typically rely on frequencies of substrings of the reads. Suffix trees are often utilized for this purpose, since they can be used to index and count the frequencies of substrings of all lengths. Existing suffix-tree based methods detect errors by identifying statistically under-represented branches (suffixes) and fix them. However, they do not refer back to the reads to put the correction in context. Since an error in a single read manifests itself at multiple nodes of a suffix tree, a read-driven approach that relies on its multiple manifestations is expected to perform better. Based on this observation, we develop an algorithm, PLURIBUS, which reconciles corrections suggested by multiple manifestations of an error using a voting scheme. We compare the accuracy of PLURIBUS in detecting and correcting errors against existing error correction techniques using simulated sequencing data. We also assess the impact of error correction on the performance of sequence assembly. Our results show that PLURIBUS corrects errors with improved precision and enables the assembler to generate longer contigs, particularly when the genome is longer, or coverage is lower. PLURIBUS is freely available at `http://compbio.case.edu/pluribus/`.

BCB '13, September 22 - 25, 2013, Washington, DC, USA

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems; J.3 [**Life and Medical Sciences**]: Biology and Genetics

General Terms

Algorithms, Experimentation

Keywords

Bioinformatics, next generation sequencing, error correction, suffix trees, sequence assembly.

1. INTRODUCTION

Next Generation Sequencing (NGS) technologies have replaced Sanger sequencing as the *de facto* standard. This shift can be attributed to the orders of magnitude increase in throughput and reduction in per-base sequencing cost of NGS. These desirable characteristics, however, come at the cost of shorter reads and increased error rates. In Sanger sequencing, error rates could be as low as one miscalled base in 100,000 bases sequenced. NGS technologies, on the other hand, are prone to miscalling bases at a rate that is orders of magnitude greater – as high as 1 in 100. While hardware advances are reducing this rate and the actual error rates experienced today may be lower than 1%, with average read lengths exceeding 100bp the expected number of errors per NGS read is close to one [9].

There are two main uses of the reads generated by NGS technologies: (i) resequencing or mapping, where the reads are aligned to a reference genome for further analysis, and (ii) *de novo* assembly, where an unknown genome is constructed entirely from the reads. In both of these cases, sequencing errors significantly increase the complexity of operations. When performing an alignment, sequencing errors can be handled by accounting for mismatches and short gaps [6]. However, it is difficult to ascertain whether these are due to sequencing errors or true variants in the underlying genome. During *de novo* assembly, sequencing errors interfere identification of overlaps between reads. In this case, sequencing errors in reads that bridge two contigs may cause the contigs to stay disjoint; alernately, they may induce spurious overlaps [11]. For these reasons, it is important to discard or correct sequencing errors prior to assembly.

1.1 Error Correction

In order to identify and correct errors in sequencing data, it is necessary to differentiate true genomic variants from sequencing errors. For *de novo* assembly, there is no ground truth to compare the data against. Fortunately, however, if there is sufficient coverage, i.e., if the expected number of reads that cover any given location on the genome is sufficiently large spectral alignment can be performed to estimate alignment to a reference genome. Spectral alignment works by approximating the set of substrings that would exist in the reference genome as the set of substrings in the read set that appear in at least a certain number of reads, and the reads are then aligned to this set of substrings to identify sequencing errors [5]. The problem of *error identification* then translates to the computational problem of identifying substrings with low frequency, since these are indicative of sequencing errors. The *error correction* problem determines modifications to erroneous substrings to transform them to high frequency substrings, which correspond to true genomic sequences.

1.2 Review of Existing Methods

Building on the spectral alignment approach to error correction, several algorithms and tools have been proposed. The major difference between the various existing correction tools is the error profile for which the tool corrects. The tools can be split into two categories tools that can correct only substitution type errors and those that can correct both substitution and indel type errors. The tools Quake [4], Shrec [8], HiTEC [3], and Reptile [10] were designed expressly for the correction of data produced by the Illumina platform, so they only correct substitution type errors. The work in [8] was extended in [7] with the tool Hybrid-Shrec which was designed to correct errors in reads that could have come from a mixture of hardware platforms, so it has the ability to correct both substitution and indel type errors.

Differences between tools in each category is largely the underlying data structure used to store and analyze substring frequency. Three data structures that are commonly used for this proces are: (i) K-mer based hashtables, (ii) suffix trees, and (iii) suffix arrays. K-mer based hash tables usually count the frequencies of all substrings of length K (K-mers) for a fixed K, and identify as errors, sets of overlapping K-mers with low frequency [4,10]. Suffix tree based methods, on the other hand, remove the dependency on a fixed parameter (K) by organizing all suffixes of all reads into a tree structure and identify as errors, low-frequency nodes with high-frequency siblings [7,8]. A third data structure, suffix array, is used as a compromise between K-mer based hash tables and suffix trees. In a suffix array, all the suffixes of the reads are still considered but they are organized into a flat data structure [3].

1.3 Contribution of Our Work

Since the suffix tree data structure is particularly expensive in terms of its space requirement, extant suffix tree based algorithms attempt to identify errors using partially constructed trees. This entails identification of errors in a *tree-driven* manner, i.e., each error is identified based on a single node of the tree. The decision on how to correct this error is based only on the siblings of that node. Note, however, that a single erroneous base in a read shows up in a large number of suffixes of the read. Consequently, each error has multiple manifestations in the suffix tree. These correlated manifestations provide important information that can be used to to improve error correction. In this paper, we argue that, at high error rates and diversity of errors (e.g., substitution errors vs. indels), conventional tree-driven methods can be significantly improved by considering all erroneous suffixes resulting from a single read error. Instead of compromising accuracy for computational efficiency, we exploit the increased availability of high-capacity computational resources to implement *read-driven* strategies that allow consideration of all manifestations of an error in guiding the correction process.

1.4 Results

We comprehensively compare the performance of our method, Pluribus against existing methods through systematic computational experiments on the human genome. For this purpose, we use a sequencing simulator, ART, which is designed to generate realistic reads, taking into account the characteristics of each existing sequencing platform [2]. We compare the performance of Pluribus and other methods in detecting and correcting sequencing errors as a function of genome length and coverage. Finally, we assess the impact of error correction on the performance of sequence assembly by comparing the performance of a state-of-the-art assembler, Velvet in three contexts: (i) using error-free reads, (ii) using uncorrected reads, and (iii) using reads corrected by Pluribus and other algorithms. Our results show that Pluribus delivers increased accuracy over competing tools and this increase is most evident for data in which there is higher ambiguity between potential corrections, such as when the errors are mixtures of substitutions, insertions, and deletions, and when the coverage of the dataset is low.

2. METHODS

In this section, we first introduce our notation and formally define the error correction problem. We then explain how a suffix-tree based data structure can be used to detect and correct sequencing errors, as is done by existing methods. Subsequently, we discuss the shortcomings of these methods, particularly in detecting and correcting short insertions and deletions (indels) and propose our method for suffix tree-based error correction using multiple occurrences, Pluribus.

2.1 Notations and Problem Formulation

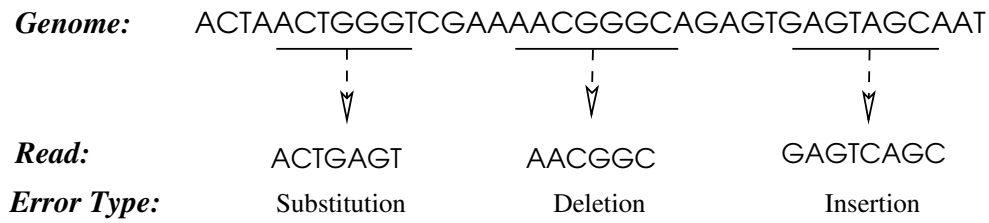


Figure 1: Illustration of the types of errors that can be encountered in sequencing.

The input to the sequencing error correction problem is a set $\mathcal{R}$ of $n = |\mathcal{R}|$ short reads from a genome G of length m (the genome length is not necessarily known). The reads are not of identical length, however, we can safely assume that the average read length, ℓ, is known. For such a set of reads, the coverage c is defined as the average number of reads that contain a given base, i.e., $c = n\ell/m$. The coverage of a sequencing run depends on multiple factors including the sequencing platform, the resources available, and the length of the genome being sequenced. Coverage levels in sequencing runs intended for *de novo* assembly typically range from 16× to more than 200× [1].

Because of the technological limitations of sequencing platforms, reads in a sequencing run contain errors; i.e., one or more bases in a read may not be identical to those at their corresponding position in G. The objective of sequencing error correction is to identify and correct such errors in reads. Various types of errors encountered in sequencing are illustrated in Figure 1. For the Illumina sequencer, one of the most commonly used sequencing platforms today, the most common type of sequencing error is the substitution error. In this type of error, a nucleotide in a read is substituted with another nucleotide. Errors can occur in the form of insertions and deletions as well, where either one or more nucleotides are inserted within a read, or one or more nucleotides are deleted from within a read. These types of sequencing errors are commonly referred to as indels.

Since substitutions are the most common form of sequencing errors, many of the existing error correction algorithms primarily target substitutions. However, indels also occur quite often, as is the case of data produced by the Roche 454 platform [9]. The presence of indels may significantly obscure the entire error correction process. During error detection, anomalies caused by indels may be interpreted as substitution errors if a substitution only algorithm is used and new errors may be introduced in an attempt to correct these misidentified errors.

The objective of the error correction problem is to identify and correct all errors (substitions, deletions, and insertions) in all of the reads in $\mathcal{R}$.

2.2 Suffix Tree Based Error Correction

Most error correction algorithms are based on the frequencies of substrings of reads. The key premise here is that, under sufficient coverage, substrings that come from an actual genomic sequence will have high frequency, whereas substrings that contain an error will have relatively low frequency. Therefore, one can identify sequencing errors by counting the frequencies of all substrings of the reads in $\mathcal{R}$ and identifying substrings that have significantly lower frequency than expected. Suffix trees provide a useful data structure for this purpose, since they can be used to track the frequency of substrings of any length by indexing all suffixes of all reads in $\mathcal{R}$. Once all suffixes are indexed, the errors manifest themselves as low-frequency subtrees in the suffix tree, whose roots have high-frequency siblings. The key idea behind suffix-tree based error correction is illustrated in Figure 2.

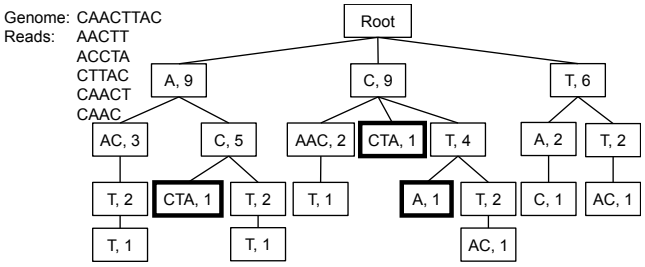


Figure 2: Illustration of the use of suffix trees to identify and correct sequencing errors. A simple hypothetical genome and a set of 5 reads from this genome are shown on the left. The second read contains a substitution error, where a T is substituted with a C. The suffix tree that indexes all suffixes of these reads is shown on the right. Each node in the tree represents the string that is obtained by reading along the path from the root to that node. The number at each node corresponds to the frequency of the respective string as a substring of the reads in the read set. The nodes highlighted in bold are those that indicate the error in the second read, since they have lower frequency compared to their siblings.

As seen in Figure 2, if the cutoff value that separates high and low frequency is 1, all of the suffixes that are incident on a sequencing error manifest themselves as low frequency nodes in the suffix tree. Siblings of those nodes also provide indicators for what the sequence may have been. One point of note is that at a certain depth in the tree, all the nodes have low frequency. Because of this, only low frequency nodes that have high frequency siblings should be considered for error correction. Analogously for a certain cutoff value all nodes above a certain depth in the tree will be high frequency. Noting these two points, there is a band in the suffix tree that provides the most information for error correction. The value of the cutoff influences the location of this informative region, where both high and low frequency nodes exist.

The Shrec [8] tool implements the process described above, but with the express purpose of correcting sequence data generated by Illumina hardware. Therefore, Shrec focuses

on correcting substitution errors only. However, since an insertion or deletion may be confused with multiple substition errors, a correction tool that ignores indels may introduce many new errors at the presence of indels. Based on this observation, HYBRID-SHREC [7] extends SHREC to correct indels in addition to substitution errors. This tool is target towards reads produced by multiple hardware platforms with diverse error profiles.

2.3 Utilizing Multiple Suffixes

The general approach to using a suffix tree for error correction, as implemented by SHREC and HYBRID-SHREC, is to detect errors on the tree, and use references back to $\mathcal{R}$ to correct the detected error. In this approach, the correction process only has a single substring in its scope at any given time. Therefore, while working from the tree, error correction decisions are made using the information from a single substring, and other manifestations of an error are not taken into account. One of the advantages of this method is that since correction is being performed based on the information from a single substring, only a fraction of the suffix tree needs to be in memory at any given time. We refer to this method of working from the tree as a "tree-driven" method. However, since the order in which the data structure is traversed is arbitrary, it is possible for this type of correction algorithm to modify the same read in multiple ways, depending on which manifestation of the error is detected first.

This can be particularly problematic when the error profile contains indels as well as substitution type errors, since the number of possible corrections that could be applied to a substring is more than doubled. This larger number of possibilities increases the liklihood of finding a possible modification to a read that exists by random chance rather than a sequencing error. Furthermore, with more possible correction types, substrings that are difficult to correct or those that have ambiguous correction paths will have their errors exacerbated.

We propose PLURIBUS, a correction algorithm that performs corrections by working from the reads. PLURIBUS considers each read one by one, and refers to the tree to find all low frequency substrings incident on the read. Subsequently, it uses a voting process to determine the correction that should be applied to the read. In this manner, PLURIBUS guarantees that the modification performed on a read is consistent for any arbitrary traversal of the underlying data structure, and for any arbitrary order of the reads in $\mathcal{R}$.

Algorithm 1 describes PLURIBUS in detail. There are aspects of the algorithm that should be noted. Variable C, which is instantiated on line 9 of Algorithm 1 allows duplicate entries. The possible corrections referenced in line 13 are pairs consisting of an index and a correction type. Here, index is the index of the first character of the low-frequency node in the read that the substring is from. This value can be obtained by using the index of the working suffix in the working read and the index of the first character of the low-frequency node in the working suffix. The correction type is one of nine possible values: delete, insert for each of the four bases, and substitute for each of the four bases. As modifications are made to the reads the counts of suffixes inside the suffix tree will change, so subsequent errors can be detected due to the change of frequencies in the tree. It is computationally prohibitive to have a fully dynamic suffix tree, one that is modified everytime a read is modified to properly represent the corrected read set. We approach this problem in the same manner as [8] and by extension [7]; the whole process is performed in rounds where up to a single correction is applied to each read before the tree is rebuilt using the modified reads.

Algorithm 1 PLURIBUS

Input: ▷ $\mathcal{R}$: Set of reads to be corrected
Output: ▷ $\mathcal{R}'$: Set of corrected reads

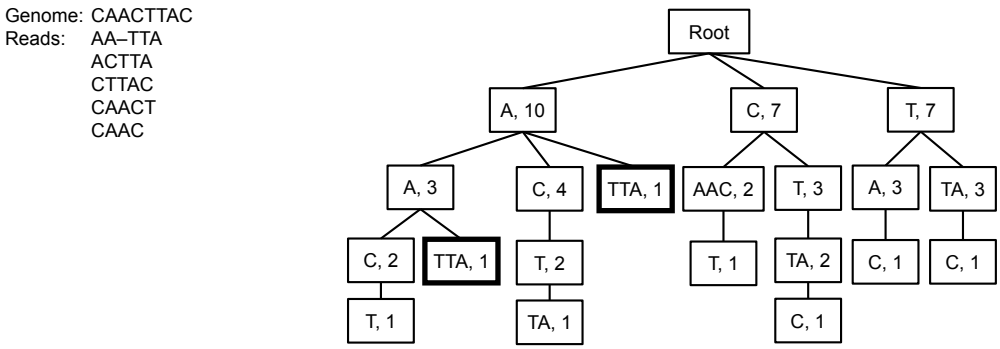
```
 1: T ← New empty suffix tree
 2: R' ← ∅
 3: for each read r ∈ R do
 4:    for each suffix s ∈ r do
 5:       Insert s into T
 6:    end for
 7: end for
 8: for each read r ∈ R do
 9:    C ← ∅
10:    for each suffix s ∈ r do
11:       Query T for s
12:       if s is incident on a low-frequency node then
13:          for each possible correction do
14:             if correction allows s to match all high-
                frequency nodes then
15:                Add correction to C
16:             end if
17:          end for
18:       end if
19:    end for
20:    MostFrequentCorrection ← most frequent item in C
21:    Apply MostFrequentCorrection to r
22:    R' ← R' ∪ r
23: end for
```

2.4 Complexity

The time to run PLURIBUS can be separated into two parts: the time to construct the suffix tree and the time to perform correction on the read set. The time to insert an arbitrary suffix into a suffix tree is bounded by the length of the suffix, for the case where each character of the suffix is incident on a unique node in the tree. This process is repeated for each suffix of each read of $\mathcal{R}$, and each read has ℓ suffixes. The time to construct the tree becomes $O(|\mathcal{R}| * \ell^2)$.

To perform correction on a single read, each suffix of the read is queried against the suffix tree. Querying the tree for a suffix takes the same amount of time as insertion of a suffix into the tree. If a query of a suffix discovers a low frequency node, each correction type is tested. However, there is a constant number of correction types, so the time to correct a read is asymptotically the same as inserting a read into the tree. So the time to perform correction is also $O(|\mathcal{R}| * \ell^2)$.

When using a *read-driven* correction method, reads are analyzed individually and independently, so only a single read needs to be in memory at a time. Thus the dominant space requirement comes from the suffix tree stored in memory. The space requirement of the suffix tree is determined by the number of nodes in the tree, the size of each node, and the number of edges in the tree. The number of edges in the tree is bounded above by 4 × number of nodes in the tree, as the size of the alphabet is 4, the characters A, C, G, and T. Both the size of a node and the size of the alphabet are constant, so the size of the suffix tree is $O(n)$, where n



a) In a *tree-driven* method the two highlighted nodes are accessed in an arbitrary order. If the tree traversal is performed in lexicographical order the lower highlighted will be hit first, and it has two possible corrections insert a C before the TTA or replace the first T of TTA with a C, both appear to be valid modifications so the *tree-driven* method has a 50% chance of performing the right correction.

b) A read driven method identifies all low frequency nodes incident on a read, so both highlighted nodes will be identified. The lower highlighted node has same two modifications that the tree-driven method found, but the upper highlighted node has a different set of possible corrections: insert an A before the TTA or insert a C before the TTA. The possible correction that appears most frequently is to insert a C before TTA, so the read driven approach will apply that correction and will deterministically correct the deletion error properly.

Figure 3: Example illustrating that (a) a *tree-driven* approach may fail to correctly identify the source of a sequencing error since it makes use of a single instance to detect and correct the error, (b) a *read-driven* approach can accurately identify and correct the actual error on the same instance.

is the number of nodes in the tree. There are a few factors that influence n, which represents the number of unique substrings present in $\mathcal{R}$. The size of the input, which is equal to $|\mathcal{R}| * \ell$, is the most influential factor. The second most influential factor is the error rate in the data. As the error rate in input increases, data begins to resemble randomly generated sequences and the number of unique substrings grows.

3. RESULTS

In this section, we compare PLURIBUS against existing error correction methods using simulated sequencing data. We first assess the performance of these methods in terms of their accuracy in detecting and correcting errors that are implanted in simulated data. Subsequently, we assess the effect of error correction on the performance of an established sequence assembly algorithm.

3.1 Error Correction Tools

We compare the performance of PLURIBUS against two state-of-the-art error correction tools: HYBRID-SHREC [7] and QUAKE [4]. We use HYBRID-SHREC in our computational experiments to represent suffix-tree based error-correction, since it is a modified version of a well-tuned tool for correcting substitution errors (SHREC) that also takes into account insertions and deletions. We also perform experiements with QUAKE, which is an optimized K-mer hash based tool that takes into account many factors, including quality factors associated with called bases.

3.2 Simulation Setup

In order to assess the accuracy of error correction, we use simulated data as the "ground truth", against which to compare our corrected reads. For this purpose, we use an established read simulator, ART [2], to generate test datasets, with properties matching realistic NGS data. ART has several hardware profiles that it can use to generate data. In the computational experiments reported in this section, we use the data characteristics of Roche 454, since reads generated by the 454 platform contains a mix of insertion, deletion, and substitution sequencing errors. Haploid subsequences ranging in length from 50K to 5M bases of human chromosome 22 reference with "N" characters removed are used as the source genomes from which ART simulated reads. Each experiment is repeated 10 times and the averages across all runs are reported here.

3.3 Precision and Recall

A direct method for comparing error correction tools against each other relies on the accuracy of correction. Using the data simulated by ART, all deviations of the reads from their respective true sequences in the source genome can be identified. Comparing the reads that have been corrected by each tool against the set of uncorrected reads allows for the identification of all the reads that have been modified and whether those modifications eliminated errors. In this context, *precision* is defined as the fraction of the of reads that are error-free after correction among all reads that are modified by the correction algorithm. On the other hand, *recall* is defined as the fraction of the reads that are error-free after correction among all reads that contain sequencing errors before correction.

To examine the robustness of each method, we consider two different properties of input read sets and, for each property, we perform experiments on a range of values that are representative of practical instances. Two important properties of the data sets that influence the error correction process are the length of the source genome and the level of coverage the reads have over that genome. Figures 4 and 5 show how the accuracy of each tool varies as a function of

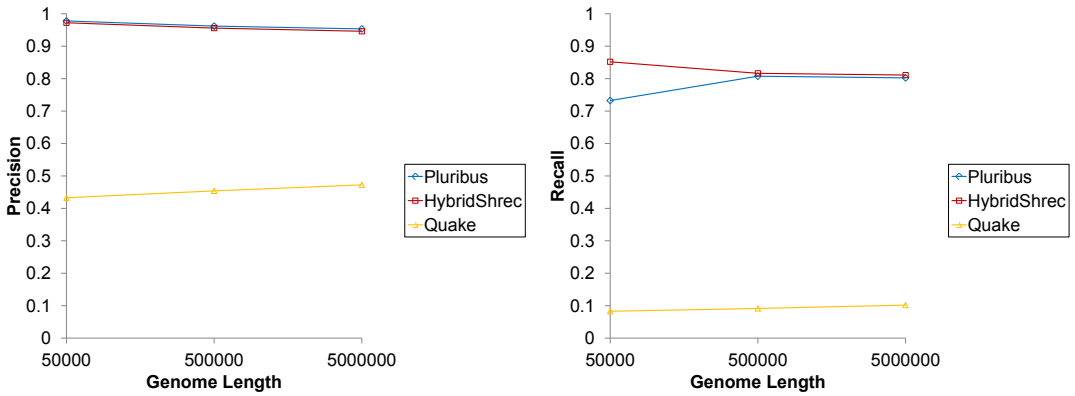


Figure 4: Precision and recall of error correction for QUAKE, HYBRID-SHREC, and the proposed algorithm PLURIBUS, as a function of genome length (ranging from 50K to 5M bases), when coverage is fixed at 15×.

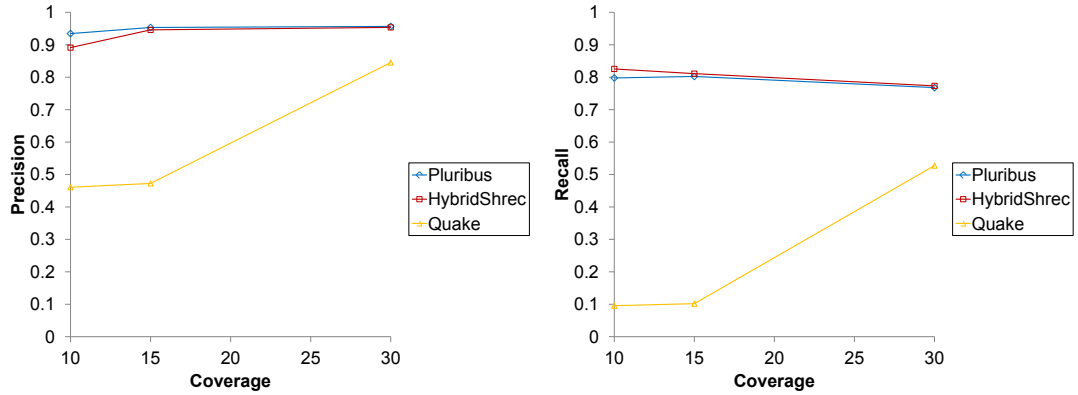


Figure 5: Precision and recall of error correction for QUAKE, HYBRID-SHREC, and the proposed algorithm PLURIBUS, as a function of coverage (ranging from 10× to 30×), when genome length is fixed at 5Mbp.

genome length and coverage.

As seen in Figure 4, the precision of error correction using suffix tree based methods, PLURIBUS and HYBRID-SHREC, is consistently high for genomes of different sizes. However, there is a slight trend showing decreased precision with increasing genome length. This is because the likelihood of generating multiple erroneous reads that coincide with each other goes up as the genome length increases. This causes ambiguities for the error correction algorithm while modifying an erroneous read. Since PLURIBUS takes into account multiple manifestations of an error, such ambiguities are better resolved. This expectation is supported by the higher precision of PLURIBUS as compared to HYBRID-SHREC for longer genomes. Based on this result, we extrapolate that the difference in the precision of PLURIBUS and HYBRID-SHREC will be larger for longer genomes. The recall of the two suffix tree methods is divergent at the 50K bp (short) genome, but it converges for longer genomes and stays consistent at around 80%. In all of the experiments, both the precision and recall of QUAKE are unsatisfactory for all genome lengths, when the coverage of the read set is 15×. From this, we conclude that QUAKE is more suitable for high coverage sequencing runs, but does not perform as well on low coverage data.

Our experiments show that, for a range of coverage levels, suffix tree based methods are able to reach high levels of accuracy, as observed in Figure 5. At coverages of 15× and 30×, PLURIBUS and HYBRID-SHREC have similar performance. However, at the lowest level of coverage tested (10×), PLURIBUS achieves higher precision at the cost of slightly reduced recall. Again, this is expected since PLURIBUS uses information from multiple manifestations of an error, which becomes particularly useful when the difference between frequencies of error-free and erroneous substrings is small. The figure also shows that QUAKE performs quite unfavorably for low levels of coverage, namely 10× and 15×, but is able to achieve significantly better accuracy for the higher coverage datasets. At 30× coverage, QUAKE is able to achieve greater than 80% precision but only slightly greater than 50% recall. This level of recall is expected, since the simulated data contains roughly one to one ratio of indels to substitution type errors and QUAKE only corrects substitution errors.

3.4 Application to Assembly

Since the end goal of reference-free error correction is to improve the performance of *de novo* assembly, we use the output of the Velvet assembler [11] to study the impact of error correction tools on the overall assembly process. For this purpose, we generate error-free and erroneous sequenc-

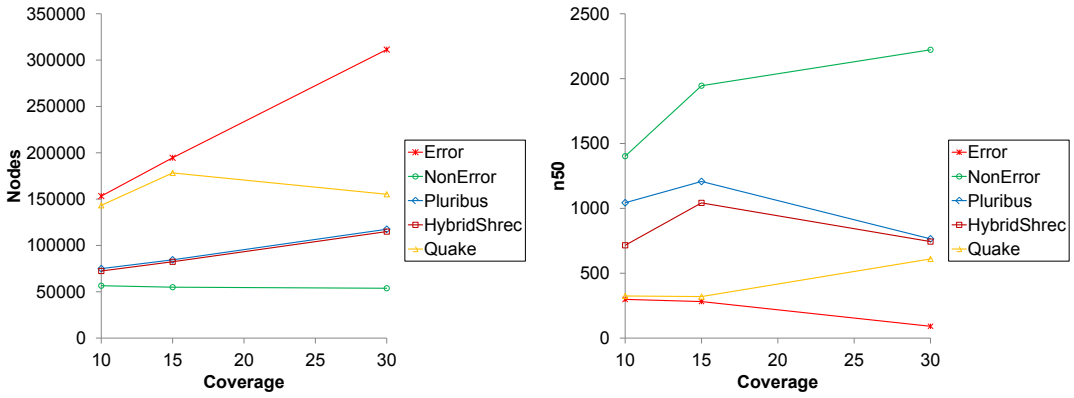


Figure 6: The assembly performance of an established assembly tool, Velvet, as a function of coverage (ranging from 10× to 30×) on a 5Mbp genome. Velvet is run on read sets that are error-free, uncorrected, and corrected by the three error correction algorithms considered here.

ing data using the ART simulator, run each correction algorithm on the erroneous data to "correct errors", and run Velvet on each of the five read sets (error-free, uncorrected, and corrected by each of the three algorithms). Since simulated data is being used, it is possible to produce datasets that contain no sequencing errors. These datasets can be used to ascertain the bounds on the improvement that error correction can have on assembly.

To compare the methods in terms of their impact on the assembly process, we use two statistics of the output produced by Velvet on each dataset: (i) the number of nodes in the final de Bruijn graph of the Velvet assembler, and (ii) the $n50$ statistic of the generated contigs. The node count indicates how much fragmentation is present in the read set. A large number of nodes in the graph is indicative of high fragmentation and is characteristic of read sets with a large number of uncorrected sequencing errors. Low node count is indicative of higher quality data, so minimizing this value is desirable. The $n50$ statistic is the weighted median of the length of the generated contigs. The ability of the assembly algorithm to generate longer contigs indicates that the set of reads available to the assembly algorithm provide information to resolve conflicts and generate larger contigs. Therefore, a larger value of the $n50$ statistic indicates better performance for the error correction algorithm.

The node count and $n50$ statistics for the performance of Velvet on the five read sets are shown in Figure 6. As can be seen in the figure, assembly performance results mimic the results from the accuracy experiments reported above. At low levels of coverage, datasets corrected by Quake have assembly performance similar to that of the datasets that were uncorrected. For a higher level of coverage, Quake shows significant improvement over the uncorrected data in both the number of nodes in the graph and in the $n50$ of the generated contigs.

Both assembly metrics show similar trends for the two suffix tree based methods, Pluribus and Hybrid-Shrec. Node counts of the two methods are close in value across the different levels of coverage, showing a linear trend with respect to coverage. A larger difference between the two methods can be seen in the $n50$ statistic. For all levels of coverage tested reads corrected by Pluribus produce contigs with the longest lengths and, as such, closest to the lengths of the contigs generated using error free data. This difference is more pronounced for the dataset with lowest (10×) coverage. This trend is identical to the trend in precision for different levels of coverage, where the difference in precision was greatest at 10× coverage.

Our results clearly suggest that, with increasing genome length and lower coverage, erroneous reads are more likely to have frequencies closer to error-free reads and error correction tools may be misled by higher frequency errors to generate more errors in an attempt to fix errors. However, if the error correction tool makes use of the information provided by multiple manifestations of the error, it is more likely to choose and make accurate corrections. As noted in the difference in the $n50$ statistics provided by Pluribus and Hybrid-Shrec, the use of multiple manifestations is particularly useful in resolving conflicts and generating longer contigs for longer genomes, particularly at shallow coverage.

3.5 Scalability and Practicality

The tool Hybrid-Shrec is the closest in design to Pluribus as they both use suffix trees for the identification and correction of sequencing errors and they target indels as well as substitution errors. Both tools perform correction in a series or rounds where up to a single error in each read is corrected in each round. To investigate the overhead of finding all manifestations of an error, we compare the timing of a single round of correction for these two tools. The computational platform we use for these computational experiments is a Dell PowerEdge R820 with 512 GB RAM and 4×8-Core Intel Xeon processors. When Hybrid-Shrec is run on the datasets where the source genome is 5Mbp, the time to perform a single round of correction takes on average 2 minutes for 10× coverage and 5.5 minutes for 30× coverage. In order to perform correction on these datasets, Pluribus requires 4.5 minutes for 10× coverage and 12.5 minutes for 30× coverage on average.

The results reported in this section are obtained by running the correction tools on data sets where the number of errors in any single read is relatively low, typically not exceeding two or three errors in a read. This low number of errors in each read allows tools such as Pluribus and Hybrid-Shrec to fully correct most reads after only two or three rounds of correction. However, if the reads are excep-

tionally long, the error rate significantly higher, or possibly both, the number of errors in a single read can become an order of magnitude or two higher. This would require these tools to be run for higher and higher number of rounds, and the round based approach would become less and less efficient. Therefore, much like most of the other existing tools, PLURIBUS is specifically designed to work best on read sets that only have up to a few errors in each read and may not scale to sequencers with longer reads and higher error rates.

4. CONCLUSION

In this paper, we propose a suffix tree based method, PLURIBUS (available at `http://compbio.case.edu/pluribus`), for correcting sequencing errors in Next Generation Sequencing (NGS) data. The key innovation of the proposed method is in the utilization of multiple manifestations of a single read for error correction. This is achieved by detecting and processing errors in a read-driven manner (as opposed to the tree-driven approach implemented by existing algorithms). Our results show that suffix trees can achieve high accuracy at low coverage levels. Furthermore, when corrections are ambiguous, i.e., in the presence of insertions or deletions, lower coverage, or longer genome length, using multiple manifestations of an error improves accuracy or error correction. This increased accuracy is also reflected in improved performance of the assembler. In future work optimizations in constructing the tree and recalling substrings can be explored to reduce the introduced overhead such as implementing the use of a suffix link algorithm. Also, our method can be tested on a rich collection of genomes, also considering longer genomes and real sequencing runs. This will enhance the generalizibility of the results reported here and establish PLURIBUS as the benchmark tool for error correction in practical applications.

Acknowledgments

We would like to thank Matthew Ruffalo (CWRU), Gökhan Yavaş (CWRU), and Wojciech Szpankowski (Purdue) for many useful discussions, and anonymous reviewers for their useful comments and suggestions. This work was supported in part by National Science Foundation (NSF) awards IIS-0916102, DBI-0835677, IOS-1124962, American Cancer Society Grant 123436-RSG-12-159-01-DMC, and by a Graduate Assistance in Areas of National Need (GAANN) fellowship to Daniel Savel from the US Department of Education.

5. REFERENCES

[1] Keith R Bradnam, Joseph N Fass, Anton Alexandrov, Paul Baranay, Michael Bechner, İnanç Birol, Sébastien Boisvert10, Jarrod A Chapman, Guillaume Chapuis, Rayan Chikhi, et al. Assemblathon 2: evaluating de novo methods of genome assembly in three vertebrate species. *arXiv preprint arXiv:1301.5406*, 2013.

[2] Weichun Huang, Leping Li, Jason R. Myers, and Gabor T. Marth. Art: a next-generation sequencing read simulator. *Bioinformatics*, 28(4):593–594, 2012.

[3] Lucian Ilie, Farideh Fazayeli, and Silvana Ilie. Hitec: accurate error correction in high-throughput sequencing data. *Bioinformatics*, 27(3):295–302, 2011.

[4] David Kelley, Michael Schatz, and Steven Salzberg. Quake: quality-aware detection and correction of sequencing errors. *Genome Biology*, 11(11):R116, 2010.

[5] Pavel A. Pevzner, Haixu Tang, and Michael S. Waterman. An eulerian path approach to dna fragment assembly. *Proceedings of the National Academy of Sciences*, 98(17):9748–9753, 2001.

[6] Matthew Ruffalo, Thomas LaFramboise, and Mehmet Koyutürk. Comparative analysis of algorithms for next-generation sequencing read alignment. *Bioinformatics*, 27(20):2790–2796, October 2011.

[7] Leena Salmela. Correction of sequencing errors in a mixed set of reads. *Bioinformatics*, 26(10):1284–1290, 2010.

[8] Jan Schröder, Heiko Schröder, Simon J. Puglisi, Ranjan Sinha, and Bertil Schmidt. Shrec: a short-read error correction method. *Bioinformatics*, 25(17):2157–2163, 2009.

[9] Jay Shendure and Hanlee Ji. Next-generation dna sequencing. *Nature biotechnology*, 26(10):1135–1145, 2008.

[10] Xiao Yang, Karin S. Dorman, and Srinivas Aluru. Reptile: representative tiling for short read error correction. *Bioinformatics*, 26(20):2526–2533, October 2010.

[11] 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.