Robust Error Correction for De Novo Assembly via Spectral Partitioning and Sequence Alignment

Andrei Alic[1], Andrés Tomás[1], José Salavert[1], Ignacio Medina[2], and Ignacio Blanquer[1]

[1] Universitat Politècnica de València
asalic@posgrado.upv.es, antodo@i3m.upv.es, josator@i3m.upv.es, iblanque@dsic.upv.es
[2] Centro de Investigación Príncipe Felipe
imedina@cipf.es

Abstract. Error correction is the first step for any de novo assembly using next generation sequencing (NGS) data. This task is quite difficult and most available error correction software only supports base mismatches. In this work we propose a novel approach based on spectral graph clustering and Smith-Waterman alignment. This approach not only supports insertions and deletions, but also do not make any assumptions about the sequenced data.

Keywords: error correction, de novo assembly, graph partition, spectral clustering

1 Introduction

De novo assembly using next generation sequencing (NGS) data crucially depends on being able to correct beforehand any errors present. NGS greatly reduced the time and cost to sequence genomes, but these advantages came with the downfall of processing huge amounts of data.

Currently, there are four big players in the sequencing market, namely Illumina, Life Technologies, PacBio and Applied Biosystems. Each equipment has distinct characteristics and, as a result, the data produced by it has different distribution of errors [15, 17, 26].

In principle, there are three possible error types for de novo assembly: substitutions, deletions and insertions. However the majority of existing error correction programs handle only the first type. This is because substitutions are far easier to correct than deletions and insertions. Furthermore, the majority of errors in data from the most popular technology (Illumina [14]) are substitutions.

Error correction programs can be categorized [25, 27] in three main groups: k-spectrum based, suffix tree/array based and sequence alignment based. Most of these programs use a k-mer decomposition of the reads produced by the sequencer. A k-mer is a specific tuple of k nucleic acid bases, where k is a parameter determined by the underlying technology and the genome of the specie being sequenced.

The first category of error correction programs analyze the distribution of k-mers among reads and use a distance (usually Hamming) to determine those reads which are closely related and, thus, aren't supposed to be different. A threshold T, firstly proposed in [1, 16], is used to categorize the k-mers in solid and insolid, by calculating their rate of appearance and then comparing this with T. Those above the threshold are deemed solid, while the others (insolid) are targeted to be converted to solid and replaced in the original locations. Some of the most well known programs in this category are Quake [9], Reptile [28], Racer [7] and Hammer [13]. All these programs can correct substitutions only.

The second group of error correction programs, the suffix tree/array based, handles multiple k values and their respective T threshold. The three most used algorithms in this category are SHREC [21], HSHREC [18] (the only one handling insertions/deletions) and HiTEC [6].

The third the group of programs employ multiple sequence alignment to search for errors, with Coral [19] and Echo [8] being two examples. The former groups those reads sharing a k-mer and afterward uses a modified version of Needleman-Wunsch to determine the consensus for a group and to modify all reads to fit together, dealing with all three error types. The latter has a similar approach but it only handles substitutions.

In this article we present a method from the multiple sequence alignment family. Like other methods in this family a two step approach is employed: first similar reads are grouped together and corrections are computed independently for each group. The novelty in our approach is the tool employed for each step: spectral clustering and Smith-Waterman alignment. Both tools have sound theoretical and practical foundations, unlike the ad hoc methods employed by most error correction programs.

The proposed method can handle all types of sequencing errors (substitutions, deletions and insertions) and it is extensible to arbitrary read lengths. Therefore, it is compatible with almost any type of technology (we plan to add support for the ABI color space in the near future). Moreover, our implementation uses a reasonable amount of CPU and memory, running in a regular desktop PC for typical experiment sizes.

2 Spectral Clustering

The first step of most error correction approaches is to compute groups of similar reads. For de novo assembly, which obviously does not have a reference genome, this is a difficult and expensive to compute task. Actually, solving perfectly the grouping problem is as hard as computing the assembly.

Most approaches for error correction use k-mers to simplify the problem while obtaining an acceptable solution. The distribution of k-mers in a genome has several properties which are exploited to identify the segments with errors and group similar reads. Usually k-mers with low frequency are considered to contain errors while k-mers with frequency close to the coverage are considered correct.

Our approach uses the number of common k-mers among reads as a measure of their similarity. This metric can be seen as an edit distance, while being much cheaper to compute than more accurate ones. Using this metric a weighted graph is built, where the nodes are each one of the n reads and the edges indicate the number of common k-mers between two reads. In contrast with other error correction programs, our approach does not make any assumptions about the k-mer distribution.

For typical experiments the k-mer graph is almost complete and its adjacency matrix does not fit in computer main memories, making inviable most clustering algorithms. However, spectral clustering combined with an iterative eigensolver does not require to explicitly build the adjacency matrix. A similar combination has been proposed for entity resolution in large databases [23].

Spectral clustering is a well known technique employed for big data analysis, machine learning and image segmentation. Using eigenvectors for clustering dates back to the original work of Fiedler [4]. Since then, this technique has been discovered and re-discovered many times in different research communities. However, we are not aware of any previous application for error correction in the genomics field. For an overview of spectral clustering, we recommend a nice survey by von Luxburg [11]. There are several clustering variants with different optimization targets. In our experience, the best heuristic for this application is the *normalized cut* proposed previously for image segmentation [22].

Spectral clustering is based on computing the second smallest eigenvector (Fiedler vector) of the Laplacian matrix

$$L = D - A$$

where A is the adjacency matrix of the graph, with each element a_{ij} equal to the weight between nodes i and j. D is a diagonal matrix with d_{ii} equal to the number of edges connecting to node i. Each d_{ii} can be easily computed by a product $A \times \mathbf{1}$, where $\mathbf{1}$ is a vector which all elements set to one. In our application, the adjacency matrix is computed implicitly using a matrix B such that

$$A = BB^T - C$$

where b_{ij} is set if the read i contains the k-mer j. C is a diagonal matrix with c_{ii} equal to the number of non-zero elements in the i-th row of B. The main advantage of this approach is that B has a large number of zero elements (sparse matrix) and can be stored efficiently in main memory.

The original formulation for the *normalized cut* heuristic proposes a generalized eigenvalue problem

$$Lx = \lambda Dx.$$

This formulation also requires to compute the second smallest eigenvalue. As all iterative solvers converge faster to the largest eigenvalues, we use instead the random walk equivalent problem

$$D^{-1}Ay = \theta y,$$

where $x = y$ and $\theta = (1 - \lambda)$ [12]. In this way, the iterative solver computes the largest eigenvalues θ and requires less iterations to converge.

The sign of each element from the eigenvector is used to partition the graph in two clusters. This process is repeated for both clusters recursively until all the reads share at least c common k-mers. However, this simple approach does not work with the intricate graph from this application. Instead, we employ a k-means algorithm to partition the vector in two clusters as proposed in [3]. The number and values of the initial points for k-means are straightforward in this case, taking the maximum and minimum element from the eigenvector. An overview of the algorithm follows:

```
function partition_group($G$)
  if common_kmers($G$) $\leq c$
    correct_group($G$)
  else
    $B \leftarrow \{B_i : i \in G\}$
    $D \leftarrow \text{diag}\left((B_G B_G^T - I)\mathbf{1}\right)$
    solve($D^{-1}(B_G B_G^T + C)y = \theta y$)
    $\{G_+, G_-\} \leftarrow$ kmeans_split($y$)
    partition_group($G_+$)
    partition_group($G_-$)
  end
```

To compute the eigenvector y we use the Krylov-Schur [24] iterative method implemented in SLEPc [5], a parallel library developed by the authors for solving large and sparse eigenproblems. The correction algorithm applied to each computed cluster is discussed in the following section.

3 Error Correction

The next step following the partitioning comprises two phases: a more sensitive re-clustering of the reads in a group generated by the partitioning mechanism and the actual error correction.

The first phase uses a modified version of the Smith-Waterman algorithm to achieve a more sensitive split of a group in subgroups. The variant employed in our algorithm accepts gaps at the beginning and the end of the compared items. A local alignment method is preferred over a global one because the whole purpose of the method is to identify the longest common chunk (the overlap) and not to see the overall differences between the full reads. Furthermore, for each subgroup, a special data structure called consensus of the reads is also generated. A consensus is a special data structure which holds the entire information for a subgroup like the distribution of bases per column, the id of the reads on a column, the original base at a certain position and the most representative element for a column. The initial consensus is built when the corresponding subgroup is established, using the first read as a seed for the columns. When a read is compared with a subgroup, the Smith-Waterman algorithm aligns the information from each column of the consensus with the data in the read. If the alignment has a number of errors under a threshold T, the comparison is valid and the result of the alignment is added to the consensus. An example of the consensus can be seen in the lower half of Table 1. At the end of this step, all reads from the initial group are divided in subgroups and their bases are kept in the consensus.

The second phase, the actual error correction, traverses the list of subgroups and, when they fit a certain size, tries to correct them by using the distribution of bases for each column. While the algorithm loops over the columns in the consensus, the reads are recreated with the nucleotide deemed as right for each position. When a correction cannot be made for a certain column, the original bases from their original positions are used instead. Given the fact that the consensus is the result of an alignment (basically a contig formed by the overlapping reads), there are a number of cases which can be encountered and must be handled for each column. Using the data from Table 1 as an example, the following cases can be deduced:

Column 4: Not enough bases to do the correction, keep original.
Column 5: $2 \times C, 1 \times T \Rightarrow$ The majority C is considered to be correct.
Column 6: $1 \times G, 1 \times C, 1 \times T \Rightarrow$ No clear winner, keep the original bases.
Column 12: $4 \times T, 1 \times N \Rightarrow$ The unknown base must be a T
Column 18: $4 \times -, 1 \times T \Rightarrow$ Insertion present only in read 4, delete the corresponding position from the consensus.
Column 22: $4 \times N, 1 \times A \Rightarrow$ The correct base is considered to be A.
Column 29: $2 \times C, 2 \times - \Rightarrow$ Equal coverage, but 'C' is chosen because has more weight than a missing value.

Table 1. Correction example

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
1					C	G	G	T	A	A	G	T	A	T	G	A	A	-	C	A	T	A	C	A	T	C	G	A	-	T	G	T				
2									A	A	G	T	A	T	G	A	A	-	C	A	T	N	C	A	T	C	G	A								
3	G	G	C	A	C	T	G	T	A	A	G	N	A	T	C	A	A	-	C	A	T	N	C	A	T	C	G	A	-	T	G	T				
4										A	G	T	A	T	G	A	A	T	C	A	T	N	C	A	T	C	G	A	C	T	G	T	G	G	A	A
5				G	T	C	G	T	A	A	G	T	A	T	G	A	A	-	C	A	T	N	C	A	T	C	G	A	C	T	G	T	G	G		
-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0
N	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
A	0	0	0	1	0	0	0	0	4	5	0	0	5	0	0	5	5	0	0	5	0	5	0	5	0	0	0	5	0	0	0	0	0	0	1	1
C	0	0	1	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	5	0	0	0	5	0	0	5	0	0	2	0	0	0	0	0	0	0
G	1	1	0	1	1	1	3	0	0	0	5	0	0	0	4	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	4	0	2	2	0	0
T	0	0	0	0	1	1	0	3	0	0	0	4	0	5	0	0	0	1	0	0	5	0	0	0	5	0	0	0	0	4	0	4	0	0	0	0

Finally, all reads, whether they are corrected or not, are output to a file in *fasta* format. The original information from the input file for each record is kept, but an additional field marking reads as being corrected or not is added. In this way, the results of the correction process can be used as an input for other programs, such as a genome assembler.

4 Results and Discussion

We present the results of our algorithm running on two real datatsets. As a reference organism, we chose E Coli K12 MG1655 with a genome length of 4639 kb. This reference has a high quality assembly and it is used in previous error correction literature as a benchmark [19, 27, 28, 6, 7]. Since our algorithm was designed from the ground up with indels in mind and the Illumina sequencers mostly generate substitutions errors, we only tested with data from Roche 454. Table 2 contains the description for the datasets used.

Table 2. Datasets

Dataset	Run ID	Platform	Read length	Number of reads
D1	SRR001355	454	$50 \sim 831$	256503
D2	SRR000868	454	$56 \sim 625$	230517

The main issue when using real data for error correction benchmarking is the uncertainty of the exact location of a read. A read could be mapped multiple times in a reference genome and the mapping errors can actually be variances (SNPs). Furthermore, even the matching bases can be errors because the available assembled genomes are not the exact representation of the real data. As a result, the whole process of finding the right site for a read has a certain degree of inexactitude. To alleviate this problem, we filtered the input data to eliminate all non-mapping and multi-mapping reads. Once we cleaned the dataset, we ran our algorithm. To test it, we followed the process described in [2, 20, 28]. We mapped the original and the corrected reads on the same reference genome we used for filtering. The process was carried out using MOSAIK [10] with its default parameters. The difference between the edit distances (field NM) in the resulting alignment files, original and corrected, was used to compute the following indexes (calculated in number of bases):

- TP (true positives) existing errors, corrected
- TN (true negatives) sane bases, not modified
- FP (false positives) sane bases wrongly considered being faulty
- FN (false negatives) erroneous bases considered being correct

One of the best metrics available to evaluate the error correction process is the *gain* which tells the percentage of errors fixed by an algorithm. The formula used by previous works [19, 27] is

$$G = \frac{TP - FP}{TP + FN}.$$

The results obtained with the test data are shown in Table 3. These results were generated using a kmer of length 8 and allowing 30 errors.

Table 3. Results

Dataset	TP	FP	FN	Gain
D1	234257	139208	739828	0.097
D2	544374	60234	721324	0.382

Lastly, an important aspect of the error correction step is the resource consumption of the application. As the quantity of data increases and the sequenced organisms have larger genomes, the need of versatile solutions is in high demand. In our case, the algorithm is bound to require more time than other solutions due to the nature of errors supported. For instance, while for Illumina a simple Hamming distance would do when comparing two reads, when it comes to indels, a more computationally expensive alternative is needed (like Smith-Waterman in our case). The solution we present herein is made up of two parts: the group generation and the actual error correction process. The former requires a memory size sufficiently big to store all the k-mers, approximately one 32 bit integer per each base in the input sequences. The error correction process requires a lot less memory than the group generation, because it stores the consensus data structure for only one group of reads at a time. Furthermore, the time spent by this step is mainly influenced by the quality of the groups. The less subgroups SW creates, the faster the algorithm is.

5 Summary and Future Work

The approach presented in this paper allows us to correct data for de novo assembly, without any assumptions about the target genome. Insertions and deletions can be corrected in addition to mismatches, supporting a great variety of sequencing platforms. Our algorithm is very robust against false corrections. If a read maps to the genome reference, there is a very high probability that its corrected version will map with less mismatches. Using an desktop PC (Intel i7 3930K CPU @ 3.2 GHz) our current development version can about one million bases in less than an hour.

Some features we plan to add in the future are: support for fastq quality score in the correction algorithm, more cases to be handled for error correction, differentiated support the existing sequencing technologies, faster, linear algorithm to handle datasets when only substitutions are allowed. Furthermore, if the input file is fastq, we plan to output the same format, with quality scores calculated by the correction mechanism. Also, we plan to implement a MPI parallel version allowing to correct very large number of reads without memory limitations.

References

1. Chaisson, M., Pevzner, P., Tang, H.: Fragment assembly with short reads. Bioinformatics 20(13), 2067–2074 (2004)
2. Chaisson, M.J., Pevzner, P.A.: Short read fragment assembly of bacterial genomes. Genome research 18(2), 324–330 (2008)
3. Dhillon, I.S.: Co-clustering documents and words using bipartite spectral graph partitioning. In: Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 269–274. ACM (2001)
4. Fiedler, M.: Algebraic connectivity of graphs. Czechoslovak Mathematical Journal 23(2), 298–305 (1973)
5. Hernandez, V., Roman, J.E., Vidal, V.: SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. ACM Trans. Math. Software 31(3), 351–362 (2005)
6. Ilie, L., Fazayeli, F., Ilie, S.: HiTEC: accurate error correction in high-throughput sequencing data. Bioinformatics 27(3), 295–302 (2011)
7. Ilie, L., Molnar, M.: RACER: Rapid and accurate correction of errors in reads. Bioinformatics 29(19), 2490–2493 (2013)
8. Kao, W.C., Chan, A.H., Song, Y.S.: ECHO: a reference-free short-read error correction algorithm. Genome research 21(7), 1181–1192 (2011)
9. Kelley, D.R., Schatz, M.C., Salzberg, S.L., et al.: Quake: quality-aware detection and correction of sequencing errors. Genome Biol 11(11), R116 (2010)
10. Lee, W.P., Stromberg, M., Ward, A., Stewart, C., Garrison, E., Marth, G.T.: Mosaik: A hash-based algorithm for accurate next-generation sequencing read mapping. arXiv preprint arXiv:1309.1149 (2013)
11. Luxburg, U.: A tutorial on spectral clustering. Statistics and Computing 17(4), 395–416 (2007)
12. Maila, M., Shi, J.: A random walks view of spectral segmentation. In: AI and STATISTICS (AISTATS) 2001 (2001)
13. Medvedev, P., Scott, E., Kakaradov, B., Pevzner, P.: Error correction of high-throughput sequencing datasets with non-uniform coverage. Bioinformatics 27(13), i137–i141 (2011)
14. Metzker, M.L.: Sequencing technologiesthe next generation. Nature Reviews Genetics 11(1), 31–46 (2009)
15. Nakamura, K., Oshima, T., Morimoto, T., Ikeda, S., Yoshikawa, H., Shiwa, Y., Ishikawa, S., Linak, M.C., Hirai, A., Takahashi, H., et al.: Sequence-specific error profile of illumina sequencers. Nucleic acids research 39(13), e90–e90 (2011)
16. Pevzner, P.A., Tang, H., Waterman, M.S.: An eulerian path approach to dna fragment assembly. Proceedings of the National Academy of Sciences 98(17), 9748–9753 (2001)
17. Ross, M.G., Russ, C., Costello, M., Hollinger, A., Lennon, N.J., Hegarty, R., Nusbaum, C., Jaffe, D.: Characterizing and measuring bias in sequence data. Genome Biol 14(5), R51 (2013)
18. Salmela, L.: Correction of sequencing errors in a mixed set of reads. Bioinformatics 26(10), 1284–1290 (2010)
19. Salmela, L., Schröder, J.: Correcting errors in short reads by multiple alignments. Bioinformatics 27(11), 1455–1461 (2011)
20. Schröder, J., Bailey, J., Conway, T., Zobel, J.: Reference-free validation of short read data. PloS one 5(9), e12681 (2010)

21. Schröder, J., Schröder, H., Puglisi, S.J., Sinha, R., Schmidt, B.: SHREC: a short-read error correction method. Bioinformatics 25(17), 2157–2163 (2009)
22. Shi, J., Malik, J.: Normalized cuts and image segmentation. Pattern Analysis and Machine Intelligence, IEEE Transactions on 22(8), 888–905 (Aug 2000)
23. Shu, L., Chen, A., Xiong, M., Meng, W.: Efficient spectral neighborhood blocking for entity resolution. In: International Conference on Data Engineering 2011 (ICDE). pp. 1–12 (2011)
24. Stewart, G.: A krylov–schur algorithm for large eigenproblems. SIAM Journal on Matrix Analysis and Applications 23(3), 601–614 (2002)
25. Tahir, M., Sardaraz, M., Ikram, A.A., Bajwa, H.: Review of genome sequence short read error correction algorithms. American Journal of Bioinformatics Research 3(1), 1–9 (2013)
26. Victoria, X., Blades, N., Ding, J., Sultana, R., Parmigiani, G.: Estimation of sequencing error rates in short reads. BMC bioinformatics 13(1), 185 (2012)
27. Yang, X., Chockalingam, S.P., Aluru, S.: A survey of error-correction methods for next-generation sequencing. Briefings in bioinformatics 14(1), 56–66 (2011)
28. Yang, X., Dorman, K.S., Aluru, S.: Reptile: representative tiling for short read error correction. Bioinformatics 26(20), 2526–2533 (2010)