# 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.

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 [**?**]) are substitutions.

Error correction programs can be categorized [**?**,**?**] 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 [**?**,**?**], 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 [**?**], Reptile [**?**], Racer [**?**] and Hammer [**?**]. 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 [**?**], HSHREC [**?**] (the only one handling insertions/deletions) and HiTEC [**?**].

The third the group of programs employ multiple sequence alignment to search for errors, with Coral [**?**] and Echo [**?**] 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. 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 [**?**].

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 [**?**]. 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 [**?**]. 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 [**?**].

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 $\theta = (1 - \lambda)$ [**?**]. In this way, the iterative solver computes the largest eigenvalues $\theta$ 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 [**?**]. 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 [**?**] iterative method implemented in SLEPc [**?**], 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 created, 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 | 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 [**?**,**?**,**?**,**?**,**?**]. 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 **??** contains the description for the datasets used.

**Table 2.** Experimental Datasets

| Dataset | Oragnism | Genome Size (Mb) | Nume Reads* | Num Reads** | Technology |
|---|---|---|---|---|---|
| SRR000868 | E Coli | 4.7 | 230517 | 223055 | 454 |
| SRR363496 | T Pallidum | 1.1 | 264996 | 241560 | 454 |
| SRR520273 | T Pallidum | 1.1 | 367056 | 354052 | 454 |

The main issue when using real data is the uncertainty of the exact location of a read. A read can be mapped multiple times onto a reference genome. Secondly, the so called mapping errors can actually be variances (SNPs). Furthermore, even the matching bases can be errors because the available assembled genomes are not perfect. As a result, the whole process of finding the real location of a read is inexact. To alleviate this problem, we filtered the input data to eliminate all non-mapping and multi-mapping reads. To test our algorithm with the filtered dataset, we followed the process described in [**?**] for gapped benchmarking. The formula used by previous works [**?**,**?**] is

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

The results obtained with the test data are shown in Table **??**.

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.

**Table 3.** Results

| Dataset | Algorithm | Time Parallel $h:m:s$ 12 Threads HT* | Time Sequential $h:m:s$** | Memory $GB$ | Gain |
|---|---|---|---|---|---|
| SRR000868 | Coral | 0:02:05 | 0:11:01 | 2.7 | 0.45 |
| | HPGEc | 0:26:04/0:00:10 | 0:26:04/0:01:21 | 0.23/0.9 | 0.27 |
| | HSREC | 0:05:03 | 0:34:43 | 4.4 | -0.07 |
| SRR363496 | Coral | 0:08:03 | 0:52:40 | 2.9 | 0.56 |
| | HPGEc | 0:33:02/0:00:13 | 0:33:02/0:01:32 | 0.27/0.9 | 0.17 |
| | HSREC | 0:04:43 | 0:32:57 | 4.6 | 0.006 |
| SRR520273 | Coral | 0:15:04 | 1:29:31 | 3.2 | 0.57 |
| | HPGEc | 0:44:17/0:00:14 | 0:44:17/0:01:47 | 0.4/0.9 | 0.13 |
| | HSREC | 0:07:25 | 0:52:01 | 4.6 | -0.04 |

## 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.

Things considered for future work are: Improve the execution speed by Fine Tuning, Improve Support for Illumina (to be able to compete with Illumina-only existing Solutions), Support for Abi Color Space, Support for PacBio Native Format, Extend to Distributed Memory Implementation, Check Different Grouping Strategies.

Grid y Computación
de Altas Prestaciones

GRyCAP

# ERROR CORRECTION AND DENOVO ASSEMBLY OF NGS DATA

**Presented by: Andrei Alic**

**Supervisor: Ignacio Blanquer Espert**

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

i3M | Instituto de Instrumentación para Imagen Molecular

# INTRODUCTION

## Next Generation Sequencing

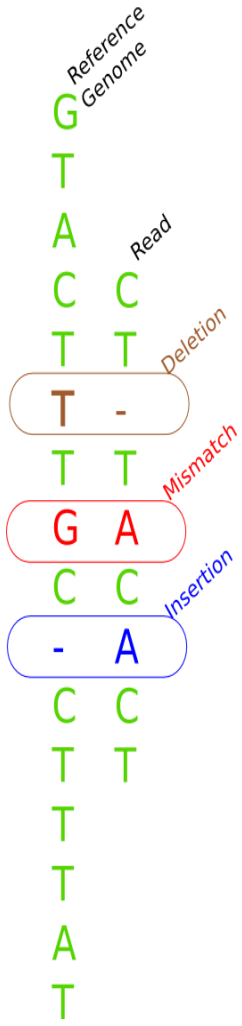


- Next (2nd) Generation Sequencing = New Generation of Instruments Capable of Reading DNA/RNA
  - Increased Through Output and Quality Compared to Sanger Method (1st gen) => Many Applications: Resequencing, deNovo Assembly, Metagenomics, Whole Population Sequencing
- Technologies: Illumina, Pacific Bioscience, Life Technologies, Roche

# INTRODUCTION

## Challenges

- Data Quantity
  - Each Locus in the Genome Sequenced Multiple Times
- Short Length of Reads
  - Datasets with 35bp Reads
- Coverage
  - Reads not Uniformly Distributed
- Errors
  - Types: Indel (Insertion, Deletion) and Mismatch



# ERROR CORRECTION ALGORITHM





- Currently Only Two Alternatives Support Indels
  - Coral (Salmela et Al 2011) and Hshrec (Schroder et Al 2010)
- Low Performance and High Memory Consumption

Our target

- Error Correction Supporting Generic Indels
- High Performance
- Technology Aware (Illumina, 454, Ion Torrent, PacBio)

# ERROR CORRECTION ALGORITHM

## Steps

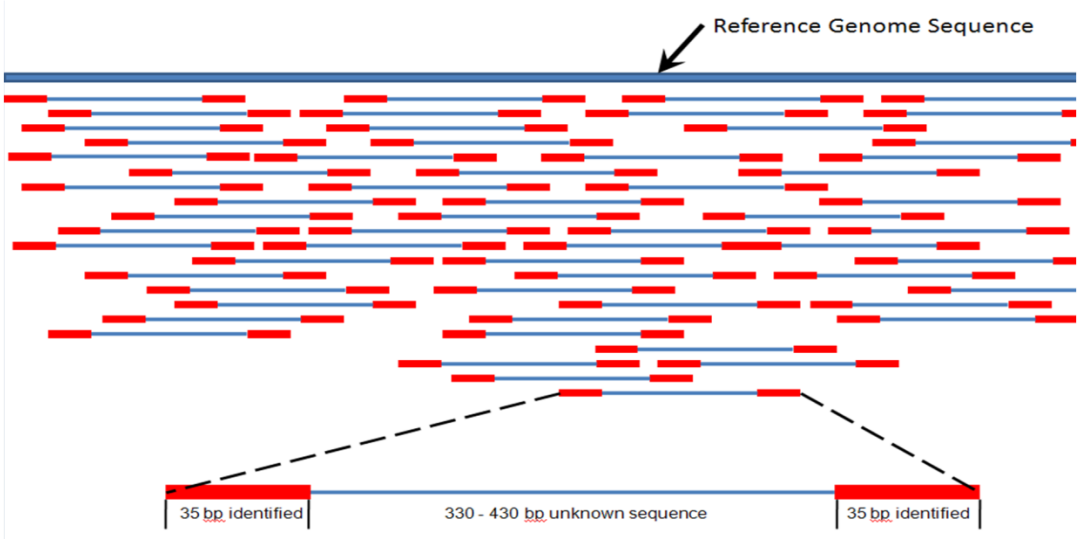


- HPGEC Has Two Distinct Steps (Implemented as Different Programs)
- First Step:
  - Grouping Using the Number of Common Kmers
    - Kmer = Fixed-Sized Segments of a Read
- Second Step:
  - Group Repartitioning using Smith Waterman Local Alignment
  - Error Correction for  Each Subgroup

# ASSEMBLY



- Use Corrected Reads to Generate the Original Genome
- Objectives:
  - Distributed, Parallel
  - High Performance When Assemble Complex Genomes
  - Technology Aware (Use Mix of Different Technologies)



# ASSEMBLY



- Our Plan:
  - Search Overlap of the Subgroups (contigs) Generated by the Error Correction Mechanism
  - Extend the Current Error Correction Mechanism to Support Distributed Processing
  - Implement Relation Between Subgroups (Reads That Can Fit in Multiple Groups)

# Thank You!