UNIVERSITAT POLITÈCNICA DE VALÈNCIA
DEPARTAMENT DE SISTEMES INFORMÀTICS I
COMPUTACIÓ

DOCTORAL THESIS

Improved Error Correction of NGS Data

Author: Andrei S. Alic
Supervisor: Dr. Ignacio Blanquer Espert

June 21, 2016

Summary

The work done for this doctorate thesis focuses on error correction of Next Generation Sequencing (NGS) data in the context of High Performance Computing (HPC).

Due to the reduction in sequencing cost, the increasing output of the sequencers and the advancements in the biological and medical sciences, the amount of NGS data has increased tremendously. Humans alone are not able to keep pace with this explosion of information, therefore computers must assist them to ease the handle of the deluge of information generated by the sequencing machines. Since NGS is no longer just a research topic (used in clinical routine to detect cancer mutations, for instance), requirements in performance and accuracy are more stringent. For sequencing to be useful outside research, the analysis software must work accurately and fast. This is where HPC comes into play. NGS processing tools should leverage the full potential of multi-core and even distributed computing, as those platforms are extensively available. Moreover, as the performance of the individual core has hit a barrier, current computing tendencies focus on adding more cores and explicitly split the computation to take advantage of them.

This thesis starts with a deep analysis of all these problems in a general and comprehensive way (to reach out to a very wide audience), in the form of an exhaustive and objective review of the NGS error correction field. We dedicate a chapter to this topic to introduce the reader gradually and gently into the world of sequencing. It presents real problems and applications of NGS that demonstrate the impact this technology has on science. The review results in the following conclusions: the need of understanding of the specificities of NGS data samples (given the high variety of technologies and features) and the need of flexible, efficient and accurate tools for error correction as a preliminary step of any

NGS postprocessing.

As a result of the explosion of NGS data, we introduce MuffinInfo. It is a piece of software capable of extracting information from the raw data produced by the sequencer to help the user understand the data. MuffinInfo uses HTML5, therefore it runs in almost any software and hardware environment. It supports custom statistics to mould itself to specific requirements. MuffinInfo can reload the results of a run which are stored in JSON format for easier integration with third party applications. Finally, our application uses threads to perform the calculations, to load the data from the disk and to handle the UI.

In continuation to our research and as a result of the single core performance limitation, we leverage the power of multi-core computers to develop a new error correction tool. The error correction of the NGS data is normally the first step of any analysis targeting NGS. As we conclude from the review performed within the frame of this thesis, many projects in different real-life applications have opted for this step before further analysis. In this sense, we propose MuffinEC, a multi-technology (Illumina, Roche 454, Ion Torrent and PacBio -experimental), any-type-of-error handling (mismatches, deletions insertions and unknown values) corrector. It surpasses other similar software by providing higher accuracy (demonstrated by three type of tests) and using less computational resources. It follows a multi-steps approach that starts by grouping all the reads using a k-mers based metric. Next, it employs the powerful Smith-Waterman algorithm to refine the groups and generate Multiple Sequence Alignments (MSAs). These MSAs are corrected by taking each column and looking for the correct base, determined by a user-adjustable percentage.

This manuscript is structured in chapters based on material that has been previously published in prestigious journals indexed by the Journal of Citation Reports (on outstanding positions) and relevant congresses.

Resumen

El trabajo realizado en el marco de esta tesis doctoral se centra en la corrección de errores en datos provenientes de técnicas de secuenciación masiva (también llamadas de nueva generación, Next Generation Sequencing o NGS) utilizando técnicas de computación intensiva.

Debido a la reducción de costes y el incremento en las prestaciones de los secuenciadores, así como en los avances en las ciencias médicas y biológicas, la cantidad de datos disponibles en NGS se ha incrementado notablemente. La utilización de computadores en el análisis de estas muestras se hace imprescindible para poder dar respuesta a la avalancha de información generada por estas técnicas. El uso de NGS transciende la investigación con numerosos ejemplos de uso clínico y agronómico (como por ejemplo la detección de mutaciones en tumores cancerosos), por lo que aparecen nuevas necesidades en cuanto al tiempo de proceso y la fiabilidad de los resultados. Para maximizar su aplicabilidad clínica, las técnicas de proceso de datos de NGS deben acelerarse y producir datos más precisos. En este contexto es en el que las técnicas de comptuación intensiva juegan un papel relevante. En la actualidad, es común disponer de computadores con varios núcleos de proceso e incluso utilizar múltiples computadores mediante técnicas de computación paralela distribuida, incluso fuera del uso científico. Las tendencias actuales hacia arquitecturas con un mayor número de núcleos (many-core) ponen de manifiesto que es ésta una aproximación relevante.

Esta tesis comienza con un análisis de los problemas fundamentales del proceso de datos en NGS de forma general y adaptado para su comprensión por una amplia audiencia, a través de una exhaustiva revisión del estado del arte en la corrección de datos de NGS. Esta revisión introduce gradualmente al lector en las técnicas de

secuenciación masiva, presentando problemas y aplicaciones reales de las técnicas de NGS, destacando el impacto de esta tecnología en ciencia. De este estudio se concluyen dos ideas principales: La necesidad de analizar de forma adecuada las características de los datos de NGS, atendiendo a la enorme variedad intrínseca que tienen las diferentes técnicas de secuenciación masiva; y la necesidad de disponer de una herramienta versátil, eficiente y precisa para la corrección de errores, como fase previa a cualquier análisis genómico.

En el contexto del análisis de datos, la tesis presenta MuffinInfo. La herramienta MuffinInfo es una aplicación software implementada mediante HTML5 para favorecer su portabilidad tanto a nivel de sistema operativo como de dispositivo. MuffinInfo obtiene información relevante de datos crudos de NGS para favorecer el entendimiento de sus características y la aplicación de técnicas de corrección de errores, soportando además la extensión mediante funciones que implementen estadísticos definidos por el usuario. MuffinInfo almacena los resultados del proceso en ficheros JSON que facilitan su integración en pipelines de proceso. Al usar HTML5, MuffinInfo puede funcionar en casi cualquier entorno hardware y software, dada el amplio soporte que tiene esta tecnología. La herramienta está implementada aprovechando múltiples hilos de ejecución y gestionando de forma concurrente el acceso a disco y la gestión del interfaz.

La segunda conclusión del análisis del estado del arte nos lleva a la oportunidad de aplicar de forma extensiva técnicas de computación de altas prestaciones en la corrección de errores para desarrollar una herramienta que soporte múltiples tecnologías (Illumina, Roche 454, Ion Torrent y experimentalmente PacBio). La herramienta propuesta (MuffinEC), soporta diferentes tipos de errores (sustituciones, delecciones, inserciones y valores desconocidos). MuffinEC supera los resultados obtenidos por las herramientas existentes en este ámbito, en los tres tipos de tests utilizados

en la tesis. Ofrece una mejor tasa de corrección, en un tiempo muy inferior y utilizando menos recursos, lo que facilita además su aplicación en muestras de mayor tamaño en computadores convencionales, donde otras herramientas no pueden funcionar por problemas de recursos. MuffinEC utiliza una aproximación basada en etapas multiples. Primero agrupa todas las secuencias utilizando la métrica de los k-mers. En segundo lugar realiza un refinamiento de los grupos mediante el alineamiento con Smith-Waterman, generando contigs resultado del alineamiento múltiple de las secuencias compatibles en el grupo. Estos contigs resultan de la corrección por columnas de atendiendo a la frecuencia individual de cada base y la aplicación de diferentes fórmulas y técnicas que facilitan discriminar errores de variantes significativas.

La tesis se estructura por capítulos cuya base ha sido previamente publicada en revistas indexadas en posiciones destacadas del índice del Journal of Citation Reports y en congresos de prestigio.

Resum

El treball realitzat en el marc d'aquesta tesi doctoral se centra en la correcció d'errors en dades provinents de tècniques de seqüenciació massiva (també anomenades de nova generació, Next Generation Sequencing o NGS) utilitzant tècniques de computació intensiva.

A causa de la reducció de costos i l'increment en les prestacions dels seqüenciadors, així com en els avenços en les ciències mèdiques i biològiques, la quantitat de dades disponibles a NGS s'ha incrementat notablement. La utilització de computadors en l'anàlisi d'aquestes mostres es fa imprescindible per poder donar resposta a l'allau d'informació generada per aquestes tècniques. L'ús de NGS transcendeix la investigació amb nombrosos exemples d'ús clínic i agronòmic (com per exemple la detecció de mutacions en tumors cancerosos), per la qual cosa apareixen noves necessitats quant al temps de procés i la fiabilitat dels resultats.

Per a maximitzar la seua aplicabilitat clínica, les tècniques de procés de dades de NGS han d'accelerar-se i produir dades més precises. En este context és en el que les tècniques de comptuación intensiva juguen un paper rellevant. En l'actualitat, és comú disposar de computadors amb diversos nuclis de procés i inclús utilitzar múltiples computadors per mitjà de tècniques de computació paral·lela distribuïda, inclús fora de l'ús científic. Les tendències actuals cap a arquitectures amb un nombre més gran de nuclis (many-core) posen de manifest que és esta una aproximació rellevant.

Aquesta tesi comença amb una anàlisi dels problemes fonamentals del procés de dades en NGS de forma general i adaptat per a la seua comprensió per una àmplia audiència, a través d'una exhaustiva revisió de l'estat de l'art en la correcció de dades de NGS. Esta revisió introduïx gradualment al lector en les tècniques de seqüenciació massiva, presentant problemes i aplicacions reals

de les tècniques de NGS, destacant l'impacte d'esta tecnologia en ciència. D'este estudi es conclouen dos idees principals: La necessitat d'analitzar de forma adequada les característiques de les dades de NGS, atenent a l'enorme varietat intrínseca que tenen les diferents tècniques de seqüenciació massiva; i la necessitat de disposar d'una ferramenta versàtil, eficient i precisa per a la correcció d'errors, com a fase prèvia a qualsevol anàlisi genòmica.

En el context de l'anàlisi de dades, la tesi presenta MuffinInfo. La ferramenta MuffinInfo és una aplicació programari implementada per mitjà de HTML5 per a afavorir la seua portabilitat tant a nivell de sistema operatiu com de dispositiu. MuffinInfo obté informació rellevant de dades crues de NGS per a afavorir l'enteniment de les seues característiques i l'aplicació de tècniques de correcció d'errors, suportant a més l'extensió per mitjà de funcions que implementen estadístics definits per l'usuari. MuffinInfo emmagatzema els resultats del procés en fitxers JSON que faciliten la seua integració en pipelines de procés. A l'usar HTML5, MuffinInfo pot funcionar en gairebé qualsevol entorn maquinari i programari, donada l'ampli suport que té esta tecnologia. La ferramenta està implementada aprofitant múltiples fils d'execució i gestionant de forma concurrent l'accés a disc i la gestió de l'interfície.

La segona conclusió de l'anàlisi de l'estat de l'art ens porta a l'oportunitat d'aplicar de forma extensiva tècniques de computació d'altes prestacions en la correcció d'errors per a desenrotllar una ferramenta que suport múltiples tecnologies (Illumina, Roche 454, Ió Torrent i experimentalment PacBio). La ferramenta proposada (MuffinEC), suporta diferents tipus d'errors (substitucions, delecciones, insercions i valors desconeguts). MuffinEC supera els resultats obtinguts per les ferramentes existents en este àmbit, en els tres tipus de tests utilitzats en la tesi. Oferix una millor taxa de correcció, en un temps molt inferior i utilitzant menys recursos, la qual cosa facilita a més la seua aplicació en mostres més gran en computadors convencionals, on altres ferramentes no po-

den funcionar per problemes de recursos. MuffinEC utilitza una aproximació basada en etapes multiples. Primer agrupa totes les seqüències utilitzant la mètrica dels k-mers. En segon lloc realitza un refinament dels grups per mitjà de l'alineament amb Smith-Waterman, generant contigs resultat de l'alineament múltiple de les seqüències compatibles en el grup. Estos contigs resulten de la correcció per columnes d'atenent a la freqüència individual de cada base i l'aplicació de diferents fórmules i tècniques que faciliten discriminar errors de variants significatives.

La tesi s'estructura per capítols la base de la qual ha sigut prèviament publicada en revistes indexades en posicions destacades de l'índex del Journal of Citation Reports i en congressos de prestigi.

To Grandpa

Declaration

I declare that the work presented herein is my own.
This work was supported by Generalitat Valenciana [GRISOLIA/2013/013].

Acknowledgements

I would like to express my special appreciation and thanks to my advisor Dr. Ignacio Blanquer Espert, whi was a tremendous mentor for me. I would like to thank you for encouraging my research and for allowing me to grow as a research scientist. Your advice on both research as well as on my career have been invaluable.

I would also like to thank my collaborators Ignacio Medina Castello, Dr. Andres Tomas Dominguez, Dr. Joaquin Dopazo and Dr. David Ruzafa. Special thanks go to Dr. Vicente Arnau Llombart, Dr. Rodica Potolea and Dr. Camelia Lemnaru for their help and guidance even before starting the PhD.

Finally, I would like acknowledge the unconditioned love of my mother Rodica Maria, grandmother Tavi, grandfather Costica (who unfortunately passed away while I still was a PhD student) and my cat Muffin. Thank you for taking great care of me and being there when I needed you the most.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

The Next Generation Sequencing (NGS) appeared around 2005 and since then its market has increased steadily, with various technologies being developed. The NGS has evolved faster than the Moore's law in Computer Science, allowing us to sequence and assemble large genomes like the Loblolly Pine with 22 Gb [3] or the Norway Spruce with 20 Gb [4] for a reasonable cost in time and resources. However, there are many other species (e.g. the Amoeba Dubia with a 670 Gb estimated genome size [5], 200x human genome's size) that are still challenging to assemble. The errors introduced by the sequencing process are one of the main reasons NGS data has to be corrected before any further use. Multiple studies have demonstrated the impact of sequencing errors on different applications of NGS, making error correction a fundamental initial step. [6, 7, 8, 9]

This chapter continues with a Motivation section where we describe why our work is necessary. Next, in Objectives, we summarise our goals for this thesis. We continue with a section dealing with the main achievements obtained from our work and conclude with the description of the organization of the whole manuscript.

Motivation

As we shall see in the chapter State of the Art, the current sequencing technologies are unable to sequence a full genome at once, even from bacteria like E. coli with a genome having 4.6Mbp. As a result, the current methods cleave the original genome in small segments which are read by a sequencer machine. To obtain the digital representation of a genome from the reads (the assembly of the genome), one must concatenate them in the right order.

The size of the genomes (3Bbp for H. sapiens) relative to the size of the reads (35bp to 90Kbp) is one real challenge that must be addressed. To allow the deduction of the right order of the

reads, each locus in the genome is sequenced multiple times. A base ends up in multiple overlapping reads. The term *coverage* denotes how many times a base on a locus in the genome has been read by the sequencer. The downside is that this process generates huge amounts of data (for example, a 400x coverage for the human genome would result in a dataset of reads having a combined 1.2 trillion bases). Unfortunately, the deluge of information cannot be processed by humans alone. The computers can process the data extremely fast, hence they can solve simple cases and mark the complex ones for later human intervention. To cut the costs, speed up the analysis and improve accuracy, the algorithms are continuously improved to handle more and more complex cases by themselves.

The errors introduced during sequencing represent another problem that must be dealt with. Overall, the research done until now strongly supports the need for error correction before any other processing of the raw data, as we shall see in chapter State of the Art. Existing error correction tools address different technologies, error types and approaches. Many error correction publications include assembly tests because one of the most important application of NGS is assembly. The assembly quality metrics (like N50) and number of assembled fragments (contigs) undoubtedly demonstrate the need of "a priori" error correction to generate a meaningful and useful output. For instance, in [10, 11], one can observe the improvement of both the performance of the assemblers and the accuracy of the results, even when using other error correction methods than the ones packaged with/embedded in the assemblers themselves. Another application of NGS is resequencing. Some papers [6, 12] specifically list this as an area where error correction has a very strong and positive impact. Finally, the short reads alignment is also improved after error correction[13, 14, 15, 16]. Variant calling is an application of NGS directly related to alignment. The errors in the reads can negatively impact the SNP

detection, because the errors may look like SNPs[9].

Objectives

The overall objective of the thesis is to advance the state-of-the-art in the field of error correction. In order to achieve this, we divided this general objective in three main objectives:

1. To lay the foundation of the thesis by looking at NGS in the context of error correction. At this step, we aim to describe succinctly the field of NGS to equip the reader with the relevant background for the actual error correction. The main accent is to be put on error correction that is to be discussed in depth and the current state-of-the-art tools are to be presented.

2. To get insights into the sequencing data. To be able to improve the existing applications, one must first understand the data and as a famous quote goes, "Knowledge is power", to us is essential to find out more about as many datasets as possible to design a tool as efficient and general as possible. Since we seek to create a whole new application, knowing what was developed previously is not enough, we must be able to generate new knowledge from newly obtained data to push the boundaries. To do that we have to be able to look into the data, therefore to develop a tool that suits our needs.

3. To correct the sequencing data. Once we have the NGS foundation, we learnt about the existing state-of-the-art and we built a customized way to look into the data, we can target the development of an effective method which fixes the errors and improves almost any further step.

These objectives are set in natural order (each built on top of the knowledge generated by the previous one). A reader with a limited knowledge in the domain can catch up and understand the work pretty easily by following the chapters in order. The accessibility to a wide audience is a secondary objective we pursue with the organization of the manuscript and the way the information is presented throughout the thesis.

Contributions

During the doctorate, we pursued a number of three main objectives. All of them have been addressed successfully as we shall see in the remainder of this section.

To address the first objective, we authored a review of stand-alone error correction methods of NGS data. Even though the title specifies error correction, the review also includes a brief, but comprehensive summary of the NGS field with the current technologies. Another unique trait is the search and inclusion model which is absent in other reviews. We specified a list of desired features for the correctors to be allowed in our review. Next, we used a reliable search approach than can be easily duplicated by the user. This same review also includes a list of scientific articles in which the stand-alone correctors were used for different purposes. Finally, we include the testing results and approaches from benchmarking-reviews, making ours a meta-review also. This work is aimed at a large audience, ranging from biologists to chemists and computer scientists. We present the information in a very accessible way with the most important background terms and information explained.

Due to the vast amount of NGS data in a typical sequencing project, a-priori information about the quality of the input and the selection of relevant information from the deluge of information can have a great impact on the project costs in human time and com-

putational resources. We introduce MuffinInfo, the materialization of the second objective, an online/offline HTML5 statistics extraction software from NGS data. One of its unique features is the ability to run on almost any software or hardware platforms. Another novel feature is its straightforward extensibility. The user is able to develop his/her own custom statistic, which can be easily integrated in the main program. Last but not least, we introduce the possibility of statistics reuse. MuffinInfo can export the results of an execution in JSON format, native to Javascript. It can open these results for later analysis. An important advantage over other similar software is the level of interactivity offered when viewing charts (which are not static images like those generated by other statistics extractors).

Once the error correction domain has been explored and we have a statistics tool customized for our needs, we proceed to the third objective with the actual error correction. MuffinEC is a parallel, indel-aware, multi-technology error corrector for NGS data. Our novel method optimizes the memory consumption and spreads the correction across multiple CPU cores. Due to its approach, it is faster and obtains better correction results than other similar software. The multiple sequence alignment correction tackles all types of errors, without resorting to simplifications like converting an unknown nucleotide to a random real base. With MuffinEC, we propose a novel, two step grouping procedure. First, the reads are grouped together by a fast, greedy mechanism to avoid comparing all reads against all reads. Next, these groups are refined. During this second step, MuffinEC may split the main group (if needed) in multiple subgroups where all the reads show strong similarity. This way, our software avoids correcting those reads clustered together at the first step as being from the same genomic locus, but, in reality, they are from distinct loci. Furthermore, this approach ensures that the misclassified reads at the greedy step are not left uncorrected.

All these three main targets are grouped as individual chapters in the manuscript.

Published Contributions

As of now, we published three articles in high impact journals and participated at two conferences:

- MuffinEC is published as "MuffinEc: Error correction for de Novo assembly via greedy partitioning and sequence alignment" in *Information Sciences*.
- The error correction review is published under the title "Objective review of de novo stand-alone error correction methods for NGS data" in *WIREs Computational Molecular Science*.
- An early version of MuffinInfo was presented as a poster at *ISMB/ECCB '15* in Dublin, Ireland. The newest feature-complete version is in press at the *Journal for Computational Biology*.
- MuffinInfo is in press as "MuffinInfo: HTML5-based statistics extractor from Next Generation Sequencing data" in *Journal of Computational Biology*.
- An early version of MuffinEC was presented at *IWBBIO '14* in Granada, Spain.

Document Organization

The thesis continues with two chapters presenting the biological and bioinformatics background and reviewing the error correction domain. They have already been published with the versions herein

being the result of the merging between the main manuscript and the Supplementary Material. The fourth chapter is published article of MuffinEC, the error corrector, the adapted for this thesis. The fifth chapter presents MuffinInfo, the statistics extractor which has been accepted for publication in a journal. We conclude with a short discussion about the contributions added with this thesis.

CHAPTER 2

State of the Art

Part of this chapter has been published as: **Alic AS.**, Ruzafa D., Dopazo J., Blanquer I. "Objective review of de novo stand alone error correction methods for NGS data". *In:* WIREs Computational Molecular Science. 2016. DOI: 10.1002/wcms.1239

There are many error correction tools in the literature that cope with different technologies and error types. However, to our knowledge, there is no complete, objective review of the modern methods that could help researchers, educators and users at the same time. There are benchmarks summarizing a number of methods, but there is none extensively focusing on the implementation, features and the overall domain (including challenges). Our work synthesises 50 *de Novo* stand-alone error-correction software.

This chapter continues with the motivation (also containing a brief description of the sequencing technologies and various error sources), followed by a presentation of the correctors. Next, section "Discussion" presents some important points related to challenges faced by correctors and how their performance is assessed in the literature. This chapter ends with some general remarks about the current state of the field of the error correction of NGS data.

We also introduce the concept of gradual recommendations. The recommendations are gradual, because they progress with the text and each one is based on the previous information. Section "Error Correction Software" includes general recommendations based on the features presented in table "Recommendations". The recommendations from section "Error correction in real projects" use as foundation the previous ones and extend the suggestions now that the reader has read about some real-world examples. Subsection "Recommendations" from section "Challenges" focuses on proposals taking into account the challenges that the correctors must address. Finally, subsection "Recommendations" from section "Testing" offers advice (now that the reader knows the methods, where these have been used and what the challenges are) based on real-world performance using different metrics.

Motivation

The market size of the next generation sequencing was estimated at $2.5 billion in 2014.[17] Furthermore, Illumina managed to lower the cost of sequencing with its HiSEQ X to ∼$1000 (in 2015) for the human genome.[18, 19] This price is quite an achievement when considering that not so long ago (2000-2003) the draft of the human genome costed about $300 million.[20] Overall, NGS has become widely used by the medical and scientific community not only for the basic biological research, but also in numerous applied fields such as medical diagnosis, forensic biology, virology and biotechnology. These are just a few clear proofs of the increasing importance of NGS in the world (not mentioning the increase in size of the segments, faster sequencing machines and improved quality of the generated data).

This quickly evolving and advancing environment facilitated the development of a myriad of methods with different applications for the NGS data. One of the most important steps (usually the first) is the correction of errors, yielding many benefits for the ulterior ones as demonstrated in sub-section "Benefits of Error Correction". Our reader may assume that an easy way to deal with the errors is to increase the coverage (i.e. add more sequencing data). While the increase in coverage indeed helps the correction process, there are still many challenges that the correctors must address (especially in *de Novo* sequencing). Furthermore, an increase in coverage comes with an increase in costs, sequencing/processing time and storage requirements.

After an extensive literature search (see approach and details in the Supplementary Material), we selected a number of 50 correctors. As one may expect, there is a tremendous amount of information scattered across these papers. We strive to summarize the deluge of information for an audience from many fields such

as bioinformatics, biology, chemistry, computer science and others with an interest in NGS. Our aim is to help the researcher in the field of error correction by grouping the information and synthesizing the existing work. Secondly, our work also comes in handy for educators because it summarizes and presents the key points of the information found in the selected articles. We tried to present the information gradually, without an abrupt and direct presentation of the correctors (our readers are not expected to have an apriori deep knowledge of the domain). Finally, the actual users of the correction software can find Table B.1 useful to choose the right tool for their specific requirements.

Sequencing Technologies

The DNA sequencing is considered to be born in 1977 with the publication of the Sanger method[21]. This method implies a large amount of DNA as template for each read and needs an independent PCR for each possible nucleotide. The PCRs are produced in presence of four deoxynucleotides and a single dideoxynucleotide, which stops the elongation. Once synthesized, the truncated DNAs are resolved by electrophoresis. During the synthesis reaction, a radioactive nucleotide (usually dATP - Deoxyadenosine triphosphate) is incorporated into the elongating strands that simplifies the determination of the sequence.

NGS methods are more efficient than Sanger sequencing in two different ways. On the one hand, in Sanger sequencing only 1 Kb (max) can be sequenced in a single experiment, whereas NGS is parallel by definition, allowing a throughput of hundreds/thousands of gigabases per run. Note that Kb, Mb and Gb are the acronyms for kilobases, megabases and gigabases. On the other hand, the chemical reactions are usually combined with the signal detection in some versions of NGS, whereas in Sanger sequencing they are two separate processes. Factors like the reduction of time, man-

Table 2.1: Information (as of November 2015) about sequencing machines as reported by the vendors themselves.

Platform	Instrument	Unit	Reads/ Unit[1]	Avg./Max Read Len. (bp)	Read Type	Quality	Error Type
Illumina	HiSeq X Ten[22]	Lane	375M	150/150	PE	≥75% bp >Q30	mismatch
	HiSeq 3000/4000[23]	Lane	312M	150/150	PE	≥75% bp >Q30	mismatch
	HiSeq NextSeq 500 High-Output[24]	Run	400M	150/150	SR/PE	≥75% bp >Q30	mismatch
	HiSeq NextSeq 500 Mid-Output[24]	Run	130M	150/150	PE	≥75% bp >Q30	mismatch
	HiSeq 2500 High-Output HISEQ SBS V4[25]	Lane	250M	125/125	SR/PE	≥80% bp >Q30	mismatch
	HiSeq 2500 High-Output TRUSEQ SBS V3[25]	Lane	186M	100/100	SR/PE	≥80% bp >Q30	mismatch
	HiSeq 2500 Rapid Run[25]	Lane	150M	250/250	SR/PE	≥75% bp >Q30	mismatch
	HiScanSQ[26]	Lane	93M	100/100	SR/PE	≥80% bp >Q30	mismatch
	Genome Analyzer IIx[27]	Lane	42M	150/150	SR/PE	≥80% bp >Q30	mismatch
	Genome Analyzer IIe[28]	Lane	42M	150/150	SR/PE	≥80% bp >Q30	mismatch
	Genome Analyzer II[29]	Flow Cell	50M	75/75	SR/PE	≥70% bp >Q30	mismatch
	Genome Analyzer[30]	Flow Cell	26M	50/50	SR/PE	≥98.5% bp error-free	mismatch
	MiSeq Reagent Kit v3[31]	Lane	25M	300/300	SR/PE	>75% bp >Q30	mismatch
	MiSeq Reagent Kit v2[31]	Lane	15M	250/250	SR/PE	>70% bp >Q30	mismatch
Thermo Fisher Scientific[32]	Ion Proton I	Chip	80M	125[a]/200	SR	NA	indel
	Ion PGM 318[33]	Chip	4M	363[a]/400	SR	>99% aligned/ measured accuracy	indel
	Ion PGM 316[33]	Chip	2M	333[a]/400	SR	>99% aligned/ measured accuracy	indel
	Ion PGM 314[33]	Chip	400K	181[a]/400	SR	>99% aligned/ measured accuracy	indel
	SOLiD 5500xl[34] W	Lane	3.2B	75/75	SR/PE/ MP	NA	mismatch

Table 2.1 Continued from previous page

Platform	Instrument	Unit	Reads/ Unit[1]	Avg./Max Read Len. (bp)	Read Type	Quality	Error Type
	SOLiD 5500 W[34]	Lane	1.6B	75/75	SR/PE/ MP	NA	mismatch
	SOLiD 5500[35, 36]	Lane	120M	75/75	SR/PE/ MP	up to 99.99% ≥QV40	mismatch
	SOLiD 5500xl[35, 36]	Lane	200M	75/75	SR/PE/ MP	up to 99.99% ≥QV40	mismatch
PacBio	RS II[37]	SMRT Cell	55K	18,181[a]/ 60,000	SR	>99.999% (QV50)	indel
	RS[38]	SMRT Cell	22k	4,575/ 20,848	SR	NA	indel
Roche 454	GS FLX Titanium XL+[39]	Run	1M	700/1000	SR	99.997% Consensus accuracy	indel
	GS FLX Titanium XLR70[39]	Run	1M	450/600	SR	99.995% Consensus accuracy	indel
	Junior[40]	Run	100K	400/-[b]	SR	99% accuracy at 400bp	indel
	Junior+[41]	Run	100K	700/-[b]	SR	99% accuracy at 700bp	indel
Oxford Nanopore	MinIon[42]	Run	4M	9,545[a,c]/ 300,000	1D/2D	96% accuracy base calling	indel
	PromethION[42]	Run	26M	9,846[a,c]/ 300,000	1D/2D	96% accuracy base calling	indel

[a]Calculated from max throughput divided by max number of reads
[b]Unspecified max length
[c]Values from Fast Mode
[1]Single Reads

power and reagents in NGS lead to lower costs making it possible to do more repeats than with the Sanger method. This results in a more accurate, reliable sequencing and better coverage. In NGS, the first step is the DNA cleavage into short fragments with lengths depending on the particular sequencer used.

In this review we focus on NGS technologies based on sequencing by synthesis (SBS), using DNA polymerase/ligase enzymes to generate a complementary strand. As defined by [43], Pacific Biosciences is the only SBS approach which has a real time sequencing strategy. All the others are synchronously controlled as we shall see in the following sub-sections where we present a brief but comprehensive description of the sequencing chemistries of the five aforementioned technologies. Fuller *et al.* also divide the methods in single molecule based (Pacific Biosciences and Oxford Nanopore - not specified in [43]) and ensemble based (Illumina, Roche 454, Ion Torrent - not specified in [43] and SOLiD). The former sequences single molecules of DNA as they are obtained from the source, while the latter relies on the amplification (cloning) of DNA fragments before starting the actual sequencing process.

The sequencing technologies explicitly supported by the methods from our review (ordered by the number of correctors supporting them) are: Illumina, Roche 454, Pacific Biosciences, Ion Torrent, Oxford Nanopore and SOLiD. Note that we put a strong accent on the actual DNA "reading" step, because this is the main step where sequencing errors are generated. For further and detailed information regarding the platforms and the entire sequencing process, please, check the Further Reading section where we listed some resources which cover the themes more in depth. Additionally, the interested reader can find more details about chemistry of SBS sequencing in [43].

Table 2.1 offers some information about a number of well-known sequencers. SR stands for single read, PE stands for pair-end reads, MP stands for mate-paired reads, 1D stands for a fragment read

in one direction and 2D stands for a fragment read both forward and reverse. Q or QV followed by a number represents the quality score for a base (depending on the reference the score can be in Phred[44] format or not)

Illumina/Solexa

In the case of Illumina sequencing, the DNA fragments receive adapters at their ends. These adapters attach themselves to the respective complementary adapters, with the latter hooked on a slide with many variants of (complementary) adapters placed on a solid surface. Next, the hooked fragments are cloned by PCR amplification, creating a spot with many copies of the same initial DNA fragment. The last step before the actual sequencing separates the complementary strands of each cloned, hooked fragment on every slide. Next, fluorescent labelled, terminated nucleotides and DNA polymerase are added as a mix (Fig. 2.1, Step 1). Fluorescent bases produce unique colours for each matching base. A polymerase is a protein that rebuilds the double helix starting from a single stranded template. It adds the complementary base for each of the template's composing nucleotides. Due to the terminated property of the free nucleotides added earlier in the mix, the polymerase attach to one and only one base per cycle (Fig. 2.1, Step 2). The sequencer registers the colour of the latest incorporated nucleotides for each slide with the cloned fragments by taking a picture (Fig. 2.1, Step 3). The process continues with the elimination of the terminator with the fluorescent label and the starting of a new cycle. The number of cycles gives the length of the read (the digital representation of a physical DNA fragment), with all reads normally having the same length. Using the snapshots, the sequencer determines the nucleotides composing a read (Fig. 2.1, Step 4).

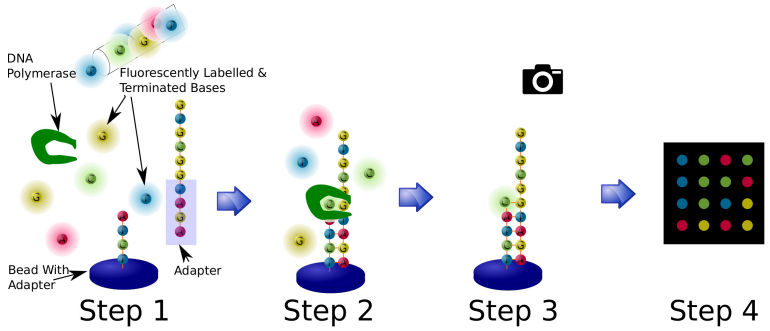


Figure 2.1: Main sequencing steps for Illumina.

Roche 454

As in Illumina's method, the 454 reads pass through a PCR amplification step and bind to adapters for which the complementaries lay hooked on a bead. Roche 454 uses the same fluorescent signalling to read the attached nucleotides. Therefore, the addition of each nucleotide releases a light signal. The main difference consists in the approach taken at each cycle. Instead of adding a solution containing all four fluorescent and terminated bases, the sequencer adds the solution with one and only one type of bases without the terminator. As a result, a variable number of bases can bind on the read at each cycle. The intensity of the signal represents the number of nucleotides added in each cycle. Roche tries to reconstruct entire homopolymers (runs of identical bases) at each cycle to save time. As an example, at Step 2 from Fig. 2.1, the sequencer adds two Cytosines (instead of one like in Illumina's case) in the depicted cycle. Generally, the sequences generated by 454 instruments have different lengths, because different numbers of bases are incorporated at each cycle.

Ion Torrent/PGM

Unlike Illumina and 454, Ion torrent sequencing is not based on the detection of optical signals. Instead, it takes advantage of the release of protons (H^+) following the addition of deoxynucleotides to the DNA strands by the DNA polymerases. The fluctuation in the pH of the solution can be easily measured and, using its level of acidity, the instrument can determine how many bases have been attached in each cycle. The bases are identified as in the case of 454.

Abi SOLiD

In this case, the reads are used to prepare clonal bead populations.[45] Instead of binding one nucleotide or a homopolymer per cycle (like the previously described sequencers), ABi uses fluorescently labelled di-nucleotide probes. Instead of individual bases, SOLiD encodes the transition between bases. At each ligation step, four DNA primers attached to different fluorescent dyes (out of the 16 DNA possibilities) are added to the reads that match to the complementary DNA primers on the bead. Next, the fluorescent part is read and afterwards cleaved from the probe. The sequencer repeats this cycle of ligation/reading/cleavage as many times as needed in order to obtain a read of a certain length. In each cycle two positions of every five are determined. Once the sequencer has executed enough cycles, it resets the template with the primer going one position backward by removing one base of the primer. In order to determine the complete sequence of the read SOLiD sequencers perform this resetting step five times. As the primer is moved one base backward, the sequencer reads each base twice, improving the robustness.

Pacific Biosciences

Pacific Biosciences uses a single-molecule real time (SMRT) sequencing approach.[46] It employs the same fluorescent labelling as the previous technologies, but instead of executing cycles of incorporating nucleotides and taking snapshots, it detects the signals in real time, as they are emitted when the incorporations occur (using a zero-mode waveguide system[47]). Like Illumina, Pacific Biosciences uses all four bases at the same time floating in the mix. It has a bead with many wells having a diameter between 70 and 100 nm, lower than the wavelength of the visible light. Due to the physical properties of the fluorescent additives, light is needed in order to make the fluorescent dye glow. Owing to this requirement, the bottom of the wells is illuminated, but due to their very small diameter, the light intensity decays exponentially along the wells, creating a shadow zone. The wells have a DNA polymerase attached to their bottoms (the illuminated zone) which rebuilds the DNA complementary strand of the DNA segments floating in the mix. Each time a nucleotide is added, the fluorescent dye is cleaved. As a result, two consecutive signals do not overlap because the sensors only record the non-cleaved fluorescent dyes as the previous ones move up in the well into the shadow zone. This approach does not require cycles, because each polymerase works independently of the others.

Oxford Nanopore

As its name suggests, this sequencing technology uses very small nanopores (allowing one nucleotide at a time) to read the DNA sequences.[48] The idea of using nanopores to decipher the DNA's code has been around since 1989, but it was not viable, until 2010 when a DNA polymerase capable of attracting the DNA to the nanopore was discovered [49]. The main components of this tech-

nology are the protein nanopores, resembling those found on the cellular membrane. These structures are inserted into an electrical resistant artificial membrane onto which an electrical potential is applied. Owing to the flowing of the potential only through the aperture of the nanopore, any molecule passing through generates a variation in the current, resulting in a specific signature. Using the previously described process, the sequencer is able to decode the DNA (or RNA or proteins). In order to get the DNA segments to pass through the hole, the segments are mixed with copies of a carrier enzyme. These carrier enzymes attach to the DNA strands. They are pulled to a nanopore where the DNA is unzipped (if necessary) and the resulting single strand passes through the aperture, producing variations in the potential. One interesting feature is the capability to sequence both strands of the DNA segment using the same nanopore, generating the so-called 2D reads (5'-3' and its complementary 3'-5' strands linked together). In order to do this, the DNA must have a hairpin structure at the end to keep the two templates together after unzipping. This way, once the first strand has passed through the hole, the complementary one is pulled through.

Errors in NGS

There are four types of basic sequencing errors: insertion, deletions, mismatches and uncalled/unknown bases (or N's).[50] The differences in the sequencing process of the aforementioned technologies lead to different types of errors. Table 2.1 lists the predominant error type for each NGS technology. This constitutes an important factor when choosing the values of the parameters for the correctors. Mismatches are prevalent in Illumina and SoLID, while indels constitute the main error type in Roche 454, Ion Torrent, Pacific Biosciences and Oxford Nanopore.[51, 52, 7, 53, 54, 55] More details about the types of errors in the aforementioned technologies and

practical experiments appear in [7, 53, 56]. The ensemble-based methods are prone to pre-sequencing errors (generated by the library preparation method and the choice of primers)[57], unless PCR-free kits are used[58, 59, 60].

Owing to its one nucleotide incorporation per cycle, the Illumina sequencers avoid insertions and deletions almost entirely.[7, 53] Sleep *et al.* [61] describe the substitution errors for various Illumina sequencers. They found that the percentage of error increases towards the 3' end of the reads [62, 9] because of a phenomenon called dephasing/phasing. It is caused by the fact that an error generated at one cycle affects the next cycles, hence the increased number of errors towards the end.[61] This same phenomenon is the main cause for the limited length of the reads in all ensemble-based SBS methods where some strands in a group of clones may fall behind resulting in a de-synchronization of the emissions of each clone in a group.[43] Another important cause of sequencing errors is the crosstalk arising due to the overlap of dye emission frequencies. The Illumina Genome Analyzer® uses a red laser to read A and C and a green laser to read G and T. As a result, the Genome Analyzer® produces many substitution G $\rightarrow$ T and C $\rightarrow$ A.[9, 63] Related to the previous cause for the MiSeq® sequencer, Schirmer *et al.* determined that A $\leftarrow\rightarrow$ C substitution errors appeared more often than G $\leftarrow\rightarrow$ T (red laser/filtering problem).[57] They also studied the relation between the position of bases in a read and the quality scores for Illumina MiSeq data. Generally, errors occurring between the start and the middle of the reads had much higher quality scores, than those in the second half of a read. Furthermore, the authors found several 3-mer motifs usually preceding substitutions and indels, resulting from the selection of primers and library design. The estimated error rate for Illumina sequencers is between 1 and 2.5%.[64, 65, 9]

Following the brief description of the Roche 454's sequencing approach, it becomes clear that, analogous to Illumina, some nu-

cleotides are misclassified. Furthermore, the exact length of the homopolymers cannot be exactly determined each time [52, 66], with the sequencer introducing:

- insertions (when recorded homopolymers are longer than real ones)
- deletions (when recorded homopolymers are shorter than the real ones)

Luo *et al.*[67] demonstrate the relation between homopolymers and their length, where pyrosequencing (Roche 454 FLX Titanium®) loses accuracy as the length of the homopolymer increases. Gilles *et al.* [68] found a chemistry-related source of errors termed the CAFIE effect (carry forward and incomplete extension). Carry forward is generated by the inability to fully clean a well (unincorporated nucleotides are not removed) after a cycle. As a result, during the next base flow nucleotides are prematurely incorporated to specific sequence combinations, hence generating noise. The incomplete extension effect appears when some DNA strands on a bead miss the nucleotides incorporation at a certain flow cycle. They must wait for the next flow cycles, but they are already out-of-phase with the other strands. The Roche 454 GS Junior® has an indel error rate of 0.38 per 100 bases.[8] Gilles *et al.* report a mean error rate of 1.07%.

The Ion Torrent sequencing approach has indels as dominant error type.[55] Bragg *et al.* also observe that insertions appear more often than deletions and that (in contrast to Illumina) indels are an order of magnitude more likely to be generated. Due to the similarity in the sequencing idea between Ion Torrent and 454, it becomes clear that homopolymers pose a problem for this technology too.[8] The sequencing accuracy of the reads steadily decreased from their start to their end.[55, 8] Loman *et al.* observed that in comparison to 454 GS Junior®, the Ion Torrent

PGM® is less accurate when dealing with homopolymers (accuracy of 60% for homopolymers with six or more bases). Furthermore, for homopolymers shorter than two bases, insertion is the main error type, but the situation changes with the increase in length of the homopolymers where deletions become the norm.[55] For long homopolymers (more than 14 bases), Ion Torrent does not generate reads at all.[69] The same study reinforces the problem with homopolymers by mentioning the inability to predict the correct number of bases for homopolymers longer than eight bases. The observed error rate is 1.78% (all types of errors) in [69], between 1.68% and 4.86% (all types of errors, depending on the used kit) with 96-97% of them being homopolymers errors in [55] and 1.5 indels per 100 bases in [8].

Pacific Biosciences generates longer reads than other sequencing technologies, but the error rate is still high.[70] The errors seem to be uniformly distributed and independent of the sequence context.[71] The same authors and [72] suggest that Pacific Biosciences is more susceptible to insertions than to deletions. Currently, the error rate for Pacific Biosciences is between 15% and 20%.[72, 73]

Oxford Nanopore is an emergent technology, generating long reads with a small and portable device (the MinION® [74]). It is still in development, but there are some publications studying the sequencing results [75, 76]. The accuracy is still low, with insertion as the predominant type of error [51]. Goodwin *et al.* report a very high error rate, between 25% and 40%.

GC Content

It is widely accepted that extreme base composition of some regions poses a problem for sequencing technologies.[7] For example the GC content (rich and poor regions) is often a source of bias and unevenness in coverage. The coverage is an extremely impor-

tant aspect of the NGS, as it is needed to successfully process the output data as we discuss in section "Low-Coverage Regions and Uniformity". The problem is even more important as the bias can be introduced during the library preparation step, before the actual sequencing process.[7] This holds true for ensemble-based SBS technologies where the amplification step (emulsion PCR or bridge amplification) generates (much) lower coverage on the very GC-rich and GC-poor regions [7]. Quail *et al.* consider that this problem appears for Ion Torrent due to its double amplification step (library and template). They managed to lower the bias by using the Kapa HiFi enzyme for the fragment amplification. Furthermore, the bias can be eliminated by using PCR-free preparation kits for Illumina[58], Ion Torrent[59] and 454[60].

Ross *et al.*[7] provide an excellent measurement of the biased caused by the GC regions. They use the genomes of four species as the correct and trusted source and compare it with the data generated by Illumina MiSeq®, Ion Torrent PGM®, Pacific Biosciences RS® and Complete Genomics®. Fig. 3 and fig. 4 from the aforementioned article depict the strong variation introduced by the these GC extreme zones.

Pacific Biosciences sequencing seems to obtain better sequencing results, because of its lack of amplification before sequencing [7], but bias still plagues this technology (slight but noticeable) when faced with genomes with GC-rich regions like S. Aureus.[69] The Pacific Biosciences RS®, like the Illumina MiSeq® and Ion Torrent PGM®, is also susceptible to dissociation of fragment ends in adapter ligation.[7] High and low GC content seems to influence Oxford Nanopore too as the coverage is more variable than in zones with a 20%-60% GC content. As a result, this extreme GC content partially motivates the lack of coverage for certain regions in the genome.[77]

Benefits of Error Correction

The most important application of error correction is in the field of genome assembly where the input data is corrected before the actual assembly. Many error correction publications include tests with assemblers and real data. Various assembly metrics demonstrate the need of error correction to generate meaningful assembly output (see subsection "Assembly" from section "Testing").[78]

A second application is re-sequencing, where multiple samples from an organism with an already known genome are sequenced. The main purpose of this operation is to compare the variability among different genomes from the same species. Another purpose is the comparison of datasets from the same organism sequenced using different technologies or sample preparation procedures. Re-sequencing indirectly uses the same metrics like gain and accuracy which compare the corrected reads against a reference genome.[6, 12]

Thirdly, the authors in [13, 14, 15, 16] stress the impact of error correction on short reads aligners. Errors are dangerous because they can cause an aligner to miss the real locus of a read in the reference genome. Furthermore, in the case of repetitive regions, a faulty read from a unique path in the genome can end up in multiple locations, provided it matches the repetitive region due to the errors.

Another affected application of NGS is the detection of SNPs. Normally, an aligner maps the reads against a reference genome to search for variants, but the errors in the reads can be misleading, increasing the total number of differences.[9] Furthermore, as the distribution of SNPs is not uniform, a region can have a high density of SNPs. Errors have a higher impact in these areas.

Additionally, there are other steps following sequencing that can benefit from error correction (e.g. identification of copy number variation or chromosomal rearrangement).[6] In conclusion al-

most any possible operation on NGS data benefit from the corrected input. Section "Error correction in real projects" lists a many real studies that used the correctors included in this review.

CHAPTER 3

Error Correction

Part of this chapter has been published as: **Alic AS.**, Ruzafa D., Dopazo J., Blanquer I. "Objective review of de novo stand alone error correction methods for NGS data". *In:* WIREs Computational Molecular Science. 2016. DOI: 10.1002/wcms.1239

This chapter presents the actual error correctors and a discussion of the most important topics in the error correction field. The exact target of a corrector is not specified in most papers, the authors normally specifying DNA reads. The benchmarks performed in the same articles contain only datasets from whole genome sequencing (WGS) projects. One exception is **PAGANtec**[93] which works with transcriptome assemblies.

Approach

Driven by the need of understanding the current status in error correction, we tackled the review included in this thesis in three stages: check existing reviews, systematic literature searching and following citations in the papers found.

There are only three reviews [1, 2, 79] published, with the latest one targeting only Illumina data. They focus on benchmarking the correctors, while this article extracts key information from the work in this field, summing up all important points deemed as important. Generally, our objective review follows the rules described for other domains in [80, 81, 82, 83].

We used search engines and literature indexes to ensure a wide coverage of the relevant work. A list of key words was identified and specific scientific searching engines were used (Google Scholar®, PubMed® and Thompson Reuters Web of Science®). We tried different terms to refine our search for the relevant papers. The final versions of the search expressions (only articles written in English were accepted) are:

- (intitle:"correct" OR intitle:"correction" OR intitle:"correcting") AND (intitle:"sequence" OR "sequences" OR intitle:"read" OR intitle:"reads") AND (intitle:"error" OR intitle:"errors") for Google Scholar®

- (((error*[Title/Abstract]) AND correct*[Title/Abstract])) AND ((sequen* OR (short NEAR read*))) for PubMed®
- ((((error*) AND correct*)) AND ((sequen* OR (short NEAR read*)))) for Thompson Reuters® with Research Domains: "SCIENCE TECHNOLOGY" and Search in: "Title"

We restricted the search to publication date between 2009 and 2014, 2009 being the year of the oldest method in [1]. The results returned by the engines were manually inspected to guarantee that the encountered work fit the inclusion criteria detailed in section 3.1.1. The third stage of our search procedure consisted of following the citations from the already discovered papers.

While the original document was prepared, we also performed a monthly literature search, to include new publications that could have appeared during the elaboration of the article. We used the Google Scholar® "Cited By" feature for this periodical check, since new methods are very likely to cite work released before.

Conditions

There is an important amount of work done in the field of error correction, with many applications having specific targets like the discrimination between sequencing errors and true bases as point mutations [84]. Others [85, 86, 87] use a reference genome to correct NGS data. Some assemblers [88] have an integrated error correction method, which cannot be used separately. Others [89, 90] have stand-alone error correction modules. We herein list the conditions the programs must fulfil to be included in the present review:

- Standalone application, published in a paper of its own. It can be a part of a suite (like an assembler, e.g. the MaSuRCA assembler [89] along with QuorUM [91]), but it must have its own separate published paper.

- Published in the last 6 years, including therefore papers published since 2009 (inclusive).
- The work should have appeared in a congress, symposium and/or journal to be accessible from an indexing service. We also include papers published on preprint servers like arxiv.org and biorxiv.org.
- No restriction on the sequencing technology, just to be deemed as NGS in the literature.
- No restriction on the rank of the journal, including all trustworthy sources.
- The correctors must use final raw sequences obtained from the sequencer, i.e. FASTQ or FASTA. Software tools using other types of input like flowgrams [92] are not considered.

Technology support

Illumina is the market leader, with a 70% market share.[17] The majority of software in our review support Illumina (and in some cases other technologies at the same time), fact that reinforces the status of the aforementioned company. The second major player is Roche with its 454 line of sequencers which, despite its shutdown of its technology in 2013, is still widely used (officially supported until 2016).[94] As a matter of fact, **Karect**, one of the most recent error correctors (2015), handles indels errors from 454. We can see an increase on the support of Pacific Biosciences, but all the current correctors rely on an additional dataset on a different technology to perform the correction. Ion Torrent is not widely supported as of now, but since the prevalent errors for this technology are indels [7], the tools handling indels should also work with it. Finally,

there is only one program that targets SOLiD color-space data, namely **HSHREC**.

In our review, we have found programs supporting more than one technology. Table 3.1 enumerates the technologies supported by all correctors (column "*Tech*"). Fig. 3.1 depicts the categories in which the correctors fall. All but one of the tools supporting only one technology work with Illumina and only target mismatch errors. **Hector** is the exception to the above rule, designed only for 454 reads, supporting indels. All Pacific Biosciences software focus only on Pacific Biosciences, but they use Illumina/454 reads for the cross-correction, therefore they are classified in a separate group.

There are several software tools handling multiple technologies which can tackle all types of errors. Our reader can determine the support for different types of errors by consulting table 3.1, columns "*N*" and "*Indel*". All programs support mismatches, therefore it is not mentioned in the aforementioned table. Some programs like **HSHREC** treat all datasets in the same manner with no special handling for different technologies (albeit **HSHREC**) has a special version which can correct colour space reads, as a different executable program). We included it in the first category because the base space version does not have a target technology.

The software with different profiles can be further divided in software using the same correction method for all technologies, but setting different values for parameters, and software with internal algorithmic modifications for a certain technology. For the first group, **Coral** is a perfect example since it uses the same algorithm to correct both Illumina and 454, but in case of Illumina the algorithm sets very high values for gaps, forcing mismatches only. **Blue** on the other hand has a flag for 454 to enable searching for homopolymers errors. **Karect** has generic support for multiple sequencing technologies, running in two modes, with indels or

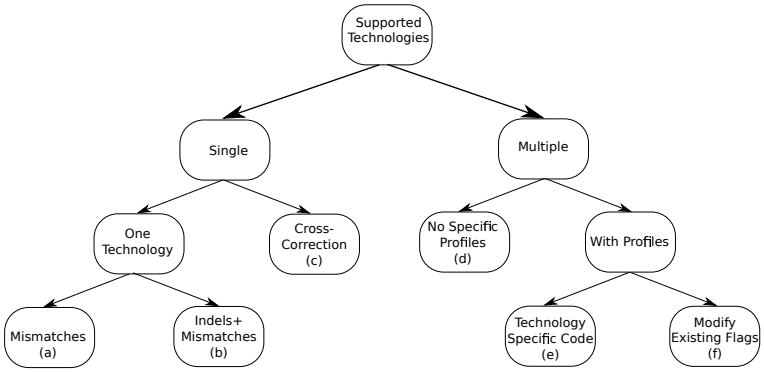


Figure 3.1: Classification using the technology support among correctors; Letters between paranthesises on the leaves used to group the algorithms in Table 3.1.

without.

A new approach is the cross-correction using a high-quality short reads dataset to correct a dataset having (much) longer, lower-quality reads. There are correctors targeting the very long reads (**LoRDEC**, **proovread**, **Jabba** and **LSC**) produced by Pacific Biosciences, 454 (**Blue**) and Oxford Nanopore (**Nanocorr**).

The latest review including software supporting indels is from 2013 [1] and it does not include the latest additions to indel-aware software, like **Blue**, **Fiona**, **Pollux** or **Karect**. The authors of the review stressed the need for better software solutions with indels support, as the results of the existing algorithms at that time (**HSHREC** and **Coral**) were not comparable to the Illumina-specific solutions.

Software Categories

We clustered the algorithms according to their core functionality, extending the work in [1, 2]. Table 3.1 summarizes some important features of the analysed software. We explain the information on the columns in the following sections.

K-Spectrum Based (ksb)

The **K**-**S**pectrum **B**ased (**ksb**) software corrects the reads employing the k-mer spectrum.[95] A k-mer is a segment from a read with k-bases. The set of all k-mers of a read is generated by using a sliding window of dimension k. At each step, the window is shifted by one element and "the visible" segment of the read is added to the spectrum set. This is by far the most popular approach, used by 28 out of 50 correctors. Generally, the applications use the k-mer spectrum (Fig. 3.2) to decide whether a k-mer is correct or not. The error-free k-mers are those appearing in a number of reads entering a predefined distribution (a Gaussian in our example). Roughly speaking, k-mers appearing in a small number of reads are considered erroneous, since the coverage is not uniform, the k-mers in the low coverage areas are under-represented (more information about k-mers in section "K-mer")

The authors of [65, 96] propose a CUDA accelerated algorithm based on spectral alignment [95, 97]. It uses a GPU-based Bloom filter to store the k-mers and to count their multiplicity (in all reads). If a k-mer appears less than a threshold m it is considered as weak, otherwise it is deemed as solid. The software first runs a voting method for each read to correct $\Delta - point$ mutations within a weak k-mer. Then, it decides to fix, trim or discard the reads with larger number of errors.

Reptile [98] employs approximate multiple alignments of segments (tiles) of reads allowing only substitutions. This software

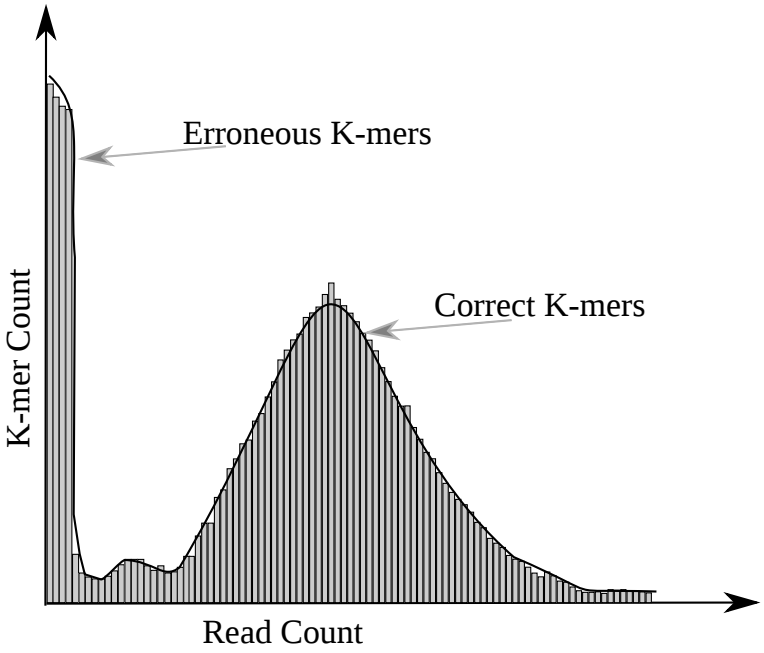


Figure 3.2: Typical distribution of k-mers used by **ksb** correctors; Vertical axis shows the number of k-mers which appear in the number of reads displayed on the horizontal axis; First peak corresponds to erroneous k-mers which appear only in a few reads; Correct k-mers typically exist in a number of reads close to the coverage; K-mers found in many reads (right part of the spectrum) typically correspond to repetitive regions.

creates these tiles by concatenating two overlapping k-mers. **Reptile** first split a read into tiles and using the k-mer spectrum it creates a Hamming graph with k-mers as nodes, connecting k-mers with a distance lower than a threshold. Starting from the first tile, it aligns the reads with tiles connected to this first tile and corrects the differences if possible.

Quake [9] uses a k-mer distribution cut-off to differentiate between trusted and untrusted k-mers. If a read contains untrusted k-mers, **Quake** searches for the set of corrections with the maximum score that convert all k-mers into trusted k-mers. The score is based on the quality values for a read coupled with the rate of nucleotide substitution. The correction process examines the changes applied to a read until it finds an unambiguous enough set of modifications that makes all k-mers trusted.

EDAR [99] removes low quality reads and, from the remaining data, calculates the coverage for all possible k-mers. Using the variable bandwidth mean-shift method [100] for each read, **EDAR** clusters the k-mers and set each cluster as erroneous or correct using a threshold derived from the normalized distribution of the coverage. The clusters a read belongs to are post-processed considering the proximity and the unusual high coverage of the k-mers. Lastly, **EDAR** detects the putative error bases for each read and splits the read at their location, creating multiple error-free fragments.

Hammer [101] is a similar method to **Reptile**, but it does not assume any uniformity distribution in the dataset. It considers the multiplicity of distinct k-mers and it finds clusters of the k-mers Hamming graph. Unlike **Reptile**, which uses consecutive tiles, **Hammer** uses spaced seeds. For each cluster Cl with more than one element, the software determines its consensus C. If C is unique, **Hammer** corrects all k-mers $k \in Cl$ with respect to C. Otherwise, it uses the quality values of each $k \in Cl$ to decide whether to keep it or to remove it k from Cl.

REDEEM [102] proposes a model to deal with errors in genomes with highly repeated content. It generates the k-mer histogram, but instead of finding errors by using the number of occurrences, it uses an Expectation Maximization method. Then, **REDEEM** estimates the expected number of appearances (attempts to read) of a k-mer, including both correct and erroneous reads. The authors propose an error model similar to **RECOUNT** [62], but instead of using full reads, they use the target k-mers.

DecGPU [103] is the first parallel and distributed error correction software for deNovo NGS data appearing in the literature. It has two separate versions, CUDA+MPI and OpenMP+MPI. It starts by building a distributed k-mer spectrum, then it removes the reads without errors and corrects the remaining (using a voting algorithm). **DecGPU** can optionally repeat the last two steps to correct reads with more than one error. Finally, the software trims/discards the erroneous reads left.

CUDA-EC2 [104] is an improved version of [65] adding the support to quality scores in the correction process. Unlike the previous version, in a preprocessing step it trims the bases with a quality score less than a threshold h. Threshold h determines also if the method could try to correct a position marked as erroneous. **CUDA-EC2** also improves the performance over the first version by using multiple concurrent threads for correcting each read in parallel.

Qamar [54] generates the k-mer frequency table t. It generates all possible 4k single-change alternatives for the k-mers with a frequency higher than a threshold, by substituting each position within with the other three possible nucleotides. Finally, **Qamar** selects the k-mer with the highest frequency in t from the alternative k-mers, and it modify the original k-mer/read accordingly.

Parallel Reptile [105] provides a parallel framework (MPI) for computing the k-mer frequency table and the Hamming distance graph. The approach divides evenly the dataset among the p pro-

cesses. Each process builds its local sorted k-spectrum. A global k-spectrum is computed merged on their completion. Finally, each process ends up with a local copy of the global k-spectrum, which is used to correct its share of reads using the same correction heuristic as **Reptile**.

BayesHammer [106] uses **Hammer**'s clustering algorithm. It improves the existing clusters through a further splitting based on the quality scores of the reads. The centres of the final subclusters are high-quality k-mers. **BayesHammer** uses them in the correction process in a majority-voting algorithm for each erroneous position in a read.

After generating the k-mer frequency table, **QuorUM** [91] splits the k-mers using a quality score threshold into reliable and not reliable. It starts the correction of the reads with its first k-mer appearing three or more times. It proceeds with the same method on both sides of the k-mer. At each step, it uses a sliding window (one base at a time) to check the existence of each newly formed k-mer in the reliable and unreliable set of reads, respectively. **QuorUM** decides at each step whether to ignore or to correct the base or trim the analysed read.

RACER [107] encodes each nucleotide into 2 bits and uses a 64-bit integer k-mer representation, converting Ns to a random real base. The method computes the occurrences of the eight possible variations for each k-mer by adding each of the possible four bases at both sides. A k-mer followed by base a is correct if its frequency is higher than a threshold t. In this case, the k-mers followed by other base b and appearing only once are corrected replacing b by a.

Musket [13] is a multi-stage corrector for Illumina datasets. Firstly, it creates the k-mer spectrum using multiple threads in a master-slave fashion and it removes the k-mers with a frequency lower than a threshold. The error correction starts classifying bases in a read as trusted (covered by at least one trusted k-mer) or

untrusted. In the latter, **Musket** searches for a unique nucleotide correction which makes all overlapping k-mers on that position trusted. Next, **Musket** tries to correct two adjacent overlapping k-mers where only one is trusted. Finally, **Musket** executes a modified voting based refinement like in [103].

Hector [52] is a modified version of **Musket** supporting 454 technology. **Hector** only handles homopolymers errors, with insertion and deletion errors in addition to mismatches. It uses of k-hopos instead of the regular k-mers used by **Musket**. A k-hopo contains k groups of adjacent homopolymers, encoding on each homopolymer its length and the nucleotide. The homopolymer encoding method helps the 454 software to obtain a bimodal spectrum analogous to the **Musket** k-mer spectrum. Unlike **Musket**, **Hector** also considers insertions or deletions as well as mismatches.

Lighter [108] proposes a three-step error correction process. First, it generates a sample set of k-mers (Bloom filter A), choosing them by a user-adjustable probability. The correct k-mers will appear more times than the erroneous ones, and thus should be kept after the sampling process. Next, it moves all trusted k-mers from A to Bloom filter B. Finally, **Lighter** corrects the reads using B and a procedure similar to [109].

HErCoOl [110] divides the k-mer frequency table into trusted and untrusted k-mers using a threshold calculated from a user input error rate. The software creates the overlap graph G of the trusted k-mers. For an untrusted k-mer of a read r, **HErCoOl** generates a set containing the closest trusted k-mers determined by the Needleman-Wunsch (NW) score. This set along with the other trusted k-mers of r mark a sub-graph in G, in which the longest path represents the corrected version of r.

Trowel [111] determines the high-quality k-mers (named bricks) using only the quality scores of their bases and stores them in an index I. Using a dual correction mechanism, **Trowel** first corrects single low quality bases followed and preceded by a brick. Sec-

ondly, it attempts to correct single low-quality nucleotides aside read-edge bricks. As low-quality bases are corrected, more candidates for correction are generated. These new candidates are included in the index *I*.

LoRDEC [71] supports PacBio but it needs an additional NGS dataset with low error rate like Illumina or 454, from which it only selects solid k-mers *skm*. **LoRDEC** creates a De Brujin Graph (DBG) with the solid k-mers. The software decompose the long PacBio reads in k-mers and tries to find them in the DBG graph. The k-mers not appearing in the DBG are weak. Reads containing no *skm* are discarded. **LoRDEC** corrects regions containing weak k-mers bordered by *skm* by finding the best path in the DBG.

BLESS [109] uses solid and weak k-mers separated by an automatic or user-adjustable threshold (M). It decomposes the reads into canonical k-mers and deems all k-mers with a multiplicity greater than M as solid. In the second phase, **BLESS** loads each read and searches for groups of overlapping solid k-mers, followed or preceded by either weak k-mers or the ends of a read (islands). If a read does not contain any island, **BLESS** searches for a solid alternative by substituting the low quality nucleotides on the first k-mer. If a valid alternative is found the correction process starts.

Blue [10] has a preprocessing step which tiles a set of reads into overlapping k-mers and outputs the distinct canonical k-mers and their multiplicity count. By using this separate step, it can perform cross correction, that is, using an Illumina dataset, it can derive the k-spectrum and use it for 454 correction. The actual error correction step tries to find a good alternative for a faulty k-mer, by searching (depth-first) the tree generated by the set of the prospective reads obtained when modifying the k-mer. The software runs recursively, that is each time **Blue** does a modification it generates all the possible alternatives for that modification onwards and starts exploring them to find the best path in the tree.

BFC[112] implements a heuristic adaptation of the method described by Chaisson *et al.*[97]. It starts by generating the hashtable of trusted k-mers using one of the two approaches: exact k-mer count with minimum frequency of three and an approximate k-mer derivation with k-mers consisting of Q20 bases. This structure aids the algorithm at detecting the longest substring s composed only from trusted k-mers, for each read. Next, **BFC** iteratively extends s at both ends and collects the possible corrections in a priority queue. This way, as the algorithm loops over a read, the past corrections influence the future ones, resulting in a correction that considers the whole structure of a read. When no s is found, **BFC** generates the set all variants with one mismatch of the first k-mer in a read. The read cannot be corrected if there are none or more than one trusted variants.

Scrible[113] targets reads obtained by sequencing intersecting pools of BACs. As a result of this sequencing approach, it is known what reads are part of what pool, therefore eliminating the problem of genome-wide repetitive regions. The algorithm generates the k-mers hashtable (forward and reverse complement) and eliminates those with very low count. After elimination, it loads each read again, generates its k-mers and search for them. If it is not found, the program tries different variants by replacing the first or the last nucleotide with the other three possibilities. **Scrible** goes k-mer by k-mer and selects correct variants for them. If at the end it ends up with multiple corrections for a read, **Scrible** iteratively tries each to apply a correction with the smallest number of nucleotide changes.

PAGANtec[93] focuses mostly on the parallelization approach of the error correction process using OpenMP. It builds a k-mer graph and tracks the errors down by observing paths supported by a low number of reads. To do this, it uses the concept of flow which is a collection of reads traversing the graph in a common manner. The authors consider two types of errors, at the ends (corrected

first) and inside of a read (corrected last), corrected in a similar manner by two different mechanisms.

ACE[114] organizes the forward and reverse-complementary K-mers of the input reads in a trie. To be able to handle very large datasets, the corrector applies a prefix-based classification based on the available memory and the size of the input data reducing the size of the trie. Next, it further reduces the memory footprint by building a root array instead of the top triangle of the trie. Finally, **ACE** employs multiple threads to build different subtries in parallel. The correction method targets the frequency of the K-mers, trying to find a high count alternative for every low-count K-mer. It is doing so by comparing the counts of K-mers given by each of the four nucleotides on a problematic position.

FADE[115] is the first FPGA-based corrector. It is based on BLESS, with several distinctions. This corrector uses a counting Bloom Filter[116] which replaces te disk based approach in BLESS. The first step is to end the reads from PC to the FPGA. The k-mers are counted in the device's memory. Once again, the reads are sent from the PC to the FPGA, where the algorithm can now determine which k-mers are weak and must be corrected. The corrected or unmodified reads are dispatched back to the PC.

Pollux[117] counts all the 31-mers for all reads. It then proceeds with a second loop over reads, but this time it generates a k-mer depth profile per read. The infrequent k-mers are not assumed as erroneous because of the low coverage regions that are correct. Instead, the program searches for discontinuity in adjacent k-mer frequencies, the k-mers with the frequencies that deviate unexpectedly.

Gu *et al.*[118] present a method using BoND-trees[119] to store the k-mers and the reads containing them on disk. The method detects suspicious bases and their positions and uses k-mers containing the bases to select the overlapping reads at that position. Utilizing a majority voting algorithm, the corrector decides if the

base is erroneous or not. A problem to the simple approach is posed by a k-mer covering a suspicious base which appears in other parts of the genomes (and it is deemed as entirely correct there). In order to mitigate this problem, the authors also verify the all the shifted k-mers containing the suspicious base, considering the fact that the probability of all k k-mers to appear twice in genome is small.

Jabba[120] is based on LoRDEC. The main idea is to align the PacBio long reads against a de Bruijn graph built with k-mers from higher quality, shorter reads. This alignment generates a path in the graph that dictates the correction. In contrast to LoRDEC, Jabba uses maximal exact matches (MEMs) to determine the path in the graph. The advantages of the MEMs as listed by the authors are the possibility of using

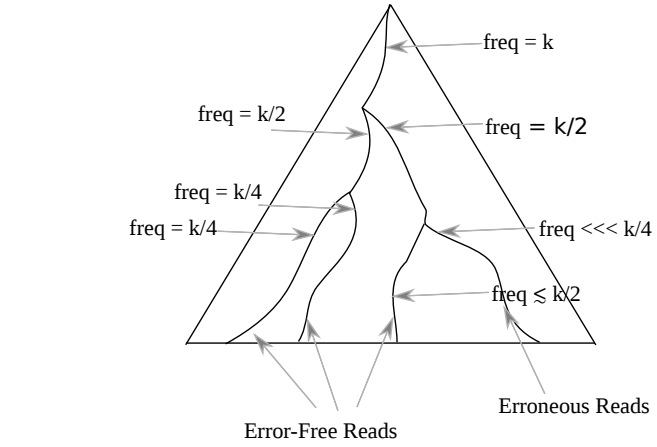
- a greater length than for k-mers
- variable length MEMs without rebuilding the whole suffix array holding them
- MEMs with different sizes than the nodes in the de Bruijn graph

Suffix Trie/Array Based (stab)

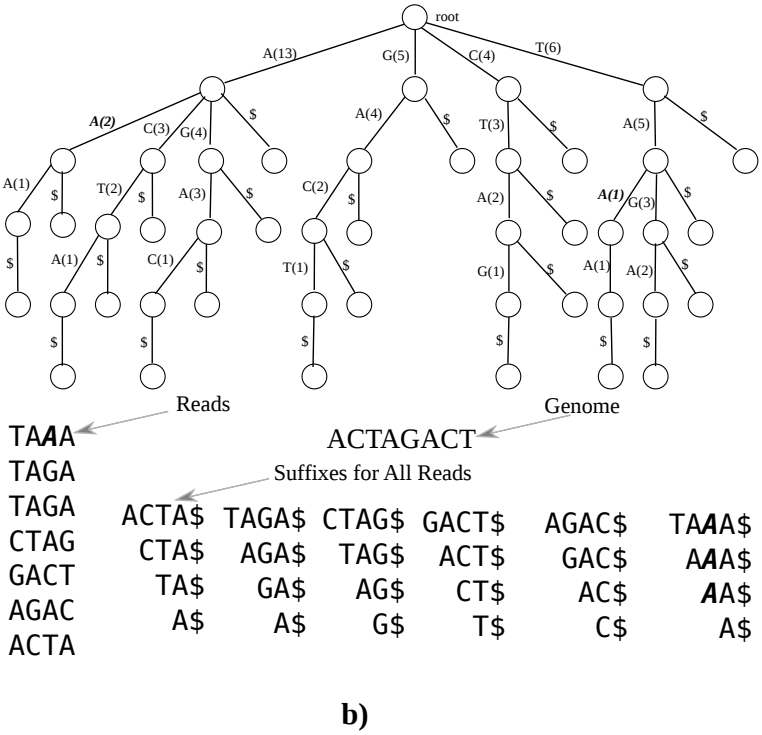
Suffix **T**rie/**A**rray **B**ased (**stab**) generally build a suffix structure with the common parts of the reads. These correctors try to locate inconsistencies in their path, while exploring the trie/array. Fig. 3.3 depicts an example where a low frequency of a divergent suffix signals a possible error case. Normally the reads on a trie follow the same path, but it happens to diverge at some point. A corrector has to decide if the split is an error or not. Fig. 3.3 a) depicts a divergence point (different nucleotide) where the frequency of one of the resulting paths is very low ($<<$ k/2) and the bases

of this path after the divergence point till the leaves are exactly the same as for the path with the frequency $\lesssim$ k/2, hence it is an error. For the other case where the frequencies are the same (k/4), a SNP (Single Nucleotide Polymorphism) causes the divergence. Otherwise, the common path till the divergence point is a repetitive region in the genome, followed by the unique region for each path. The trie in Fig. 3.3 b) with the erroneous base in bold and italic exemplifies the branching caused by an error. The \$ symbol marks the end of a suffix (a standard way of depicting suffix tries). It is crystal clear that due to the low frequency of the suffixes containing the bad base, a corrector can isolate the error and can take a valid decision given enough coverage. Given the shortness of the reads in our example we consider one base to be sufficient proof of inclusion on one branch or another. As a result, for the suffix A***A***A\$ the third base will match with its counterpart from the suffix AGA\$ (the first base is the same since we are talking about the same family of suffixes), therefore ***A*** should be G. Next, the branch TA***A***A\$ triggers a warning for the corrector due to its low frequency of its sub-branch ***A***A\$. The problematic base is again surrounded by bases that match a sibling path (i.e. TA***A***A can be converted to TAGA, with the latter having a higher frequency), therefore it is safe to assume that ***A*** is in fact G. Finally, after analysing these cases, a corrector can support its decision by detecting the relation between the suffixes A***A***A\$ and TA***A***A\$ where the former is in fact included in the latter and the corrections on both sides have an even higher degree of validity when taken together.

SHREC [64] uses a generalized suffix trie to store and fix the reads. It checks for an unbalance of edge weights between certain levels of the trie where the errors are likely to be detected. **SHREC** tries to correct a node with a smaller than expected weight by converting it to a correct sibling such that the subtree below the faulty node exactly fits in subtree rooted at the correct sibling. An



a)



b)

Figure 3.3: Suffix trie example; a) An error on the rightmost path results in branch having a very low frequency ($<<$ k/2) compared with its sibling branch ($\lesssim$ k/2); b) Example of a trie for a very short genome with read TA***A***A having an error on its third position

uncorrected node marks the read as erroneous and eliminates it from the final set of reads.

The previous software is extended in [121] to support varying length reads, indels, unknown bases and hybrid colour/base space reads. **HSHREC** finds indels like **SHREC**, but instead of converting a node to a sibling it checks if an insertion or deletion on the faulty node can make the sibling subtrees match. N's are always erroneous being either a substitution or deletion. When using colour/base space mix, the software converts all bases to colour and creates a trie in colour space. It applies the same correction method as for base space.

PSAEC [122] is an improved version of **HiTEC** based on h-order suffix arrays [123]. It also features a Pthread implementation and a reduced memory consumption. The correction algorithm implements the same steps as **HiTEC** and leading to the same results.

Like **SHREC**, **HiTEC** [6] uses weights to find erroneous bases. **HiTEC** uses suffix arrays and it uses the occurences of a witness (k-mer) u followed by a problematic base a in the pool of reads to determine the correct a. When a read cannot fit at least one u, the algorithm calculates a distribution of errors and decreases the size of u such that the segment will not lose its uniqueness (or close to) property. **HiTEC** calculates the number of iterations needed to correct a read when the count of corrected positions in a iteration drops below a threshold.

MyHybrid [12] employs both a suffix array and multiple sequence alignment for error correction. In a first stage, it detects the overlapping region (common substring) between related reads. In a second stage, the method uses the common substring (error-free) between a set of reads S as an anchor for the multiple alignment to form a consensus c for the set. Finally, **MyHybrid** uses NW to identify the differences between each read $\in S$ and the consensus c.

Pluribus [124] improves upon [121] by considering all the suffixes generated by an error instead of deciding the validity of a node based only on the siblings. **Pluribus** avoids correcting a read in multiple ways, depending on the order of data structure traversal by taking one read at a time and referencing the trie to determine low frequency segments incident on the read. It decides upon the corrections using a voting mechanism. This consideration impacts negatively in the performance, making it slower than [121].

Fiona [50] uses the same approach as **SHREC** and **HSHREC** to correct mismatches and indels. Unlike the aforementioned methods, **Fiona** optimizes the error detection by using an edit distance metric between overlapping reads. The sizes of the seeds for the alignments variate with the length of the reads and are automatically calculated for every case, extending this method in a following version named **HiTEC**. Furthermore, it accumulates the errors for a specific read and it corrects them by order after the error detection process stops, not when each individual error pops in.

Multiple Sequence Alignment Based (msab)

Multiple **S**equence **A**lignment **B**ased (**msab**) software focuses on aligning the reads to identify the overlap between them (see Fig. 3.4). The methods use different algorithms (like Needleman-Wunsch in **Coral** [15]) to build a consensus from a set of reads that are likely to fit together. Generally, these methods cluster together a number of related reads (e.g. those having at least one k-mer in common, like **Coral**), which may belong to (as the corrector may wrongly include reads from other regions) the same genomic locus. Reads containing k-mers appearing in multiple loci or erroneous k-mers matching wrong locations will normally fail in the alignment process. Being part of the same region, a **msab** corrector can generate a multiple sequence alignment and try to determine and fix the anomalies in the resulting consensus. The example Fig. 3.3 b)

demonstrates that given sufficient coverage, a corrector is able to group a number of reads, isolate the erroneous bases and make a decision if possible. In the above example, we are able to take a decision in every case. On the contrary, if we have a mismatch between the first position of the first read and the fifth position of the third read, the decision is not straightforward any more (if possible altogether). For instance, if instead of (A,A) - the correct version - the pair would be (A,N), the corrector would have to either ignore it or apply some kind of heuristic. An example of approach would be to convert N to A, since A is a valid nucleotide. On the other hand, this approach could be rendered useless by the use of quality scores where N has a very high score compared to A (the previously considered valid base may not be so valid after all). In this case it is up to the corrector to take the appropriate action using different approaches and the context of the problem.

ECHO [11] generates the k-mers and stores them along with the identifiers of the reads they belong to. Next, the method generates neighbourhoods by aligning all reads sharing a common k-mer. A valid alignment should satisfy both a score condition based on a certain fraction of errors in the overlap, and an overlap length larger than a threshold. ECHO discards any neighbourhood with a C greater than a threshold (inferred using an estimate of the expected C), avoiding corrections in highly repetitive regions. The second stage determines if a base is correct using expectation maximization. It also handles diploid genomes by considering zygosity in their expectation maximization calculus (neighbourhood finding remains unchanged). The software can automatically determine the optimal parameters.

Coral [15] starts by generating the k-mer (forward and reverse-complementary) spectrum with the list of reads appearing in. **Coral** considers each entry in the aforementioned spectrum as a neighbourhood n. The correction step is based on MSA and it starts by considering the first read in a n as the consensus C for that n.

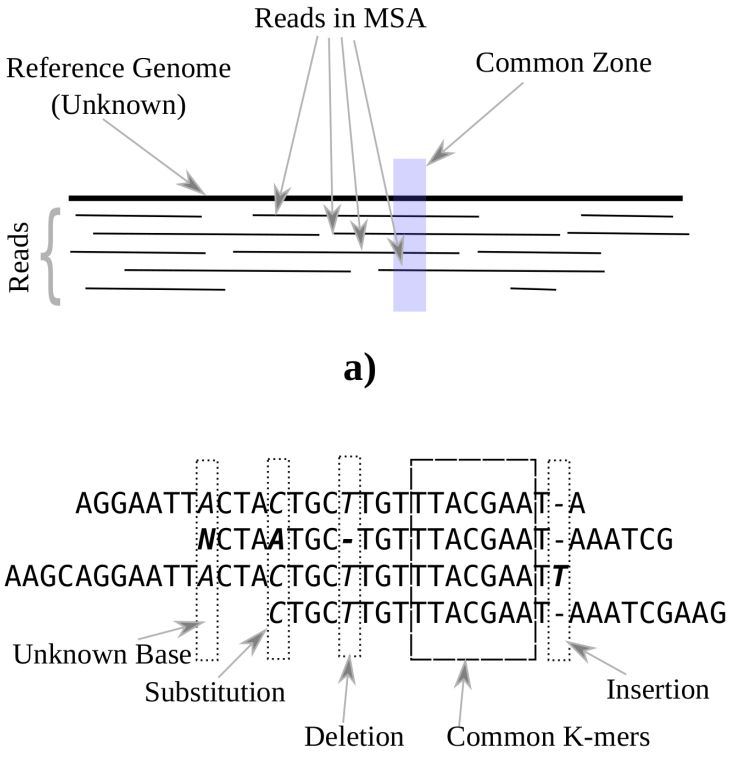


Figure 3.4: a) Multiple sequence alignment of reads versus the (prospective) reference genome; b) Example of four read with the common k-mer "TTACGAA" and the four basic types of errors.

Coral progressively adds all the other reads in n to C by using Needleman-Wunsch as a distance and updating C at each step. Finally, it corrects each column of C by considering the base frequencies and quality scores (if available).

LSC[73] performs a hybrid correction of PacBio data using a multi-steps approach. First, it compresses short (SR) and long reads (LR) by collapsing the homopolymers to a single base. Second, it selects only the high quality SRs and aligns them using Novoalign (other aligners could be used) against the concatenation of all compressed LRs. Finally, **LSC** modifies an LR using the consensus of the SRs aligning on a correction point on a LR. **LSC** outputs the decompressed LR from the left-most SR-covered point to the right-most SR-covered.

CloudRS[125] implements the ALLPATHS-LG's [126] error correction algorithm on top of Hadoop. The corrector create stacks of reads sharing a k-mer. The kmer can contain upto one wildcard (i.e. a variable base). Firstly, **CloudRS** uses a 25-mer with a central wildcard to stack the reads and correct the variable base. Next, it clusters the reads with a fixed 24-mer and fixes the bases outside the common zone. Finally, the method filters out those reads containing unique 24-mers.

Chung textitet al.[127]aim to improve the efficiency and effectiveness of **CloudRS**[125] by introducing the read-message (RM) diagram trimmed using the Gradient-number Votes (GNV) scheme. They define the RM diagram of a read as the set of the records i represented by $Arm_{left}(i)$, $Kernel(i)$, $Arm_{right}(i)$. There are as many i as k-mers in a read (denoted by R). A base at position p in a read is corrected using a variable voting scheme dependent upon p, R, k, the base number of votes and a value for fine-tuning the votes.

Like the previous PacBio correctors, **proovread**[72] uses alignments (using SHRiMP2 [128] or Bowtie2[129] - experimental support) of SR on LR. Using previous work of [130, 7] the authors

give different weight to deletions, insertions and mismatches. The application creates multiple alignments using all the short reads mapping on a small region called a bin on the a LR. A consensus of SR's resulting from a majority voting of each column of the overlapping reads decides the right bases of a bin. Chimeric regions left from the previous consensus step can be trimmed using a window-based quality filter. To increase both the performance and correction rate, **proovread** performs an iterative correction, increasing the sensitivity of the mapping process at each step.

The only corrector that handles Oxford Nanopore data in our review is **Nanocorr** [77]. Like the PacBio algorithms, it is a hybrid approach that uses Illumina MiSeq short reads to correct the long reads. The software is a Python wrapper which uses BLAST internally to align the short reads on the long Nanopore's ones as its first step. Next, Nanocorr determines the set of short read alignments to be used for correction of a long read by running a longest-increasing-subsequence dynamic algorithm. Finally, the corrector calculates a consensus (to be able to apply the corrections from the short to the long reads) using pbdagcon [1].

Karect[131] uses a partial order graph (POG) to accumulate partial alignment results. The alignments are generated for each read r from the input file which becomes the seed. To generate the MSA, **Karect** sets a size m for a group and fills it with reads with:

- an exact common k-mer
- a common k-mer with d mismatches/indels on its prefix or suffix
- two common exact sub-k-mers

[1] Accessible at https://github.com/PacificBiosciences/pbdagcon

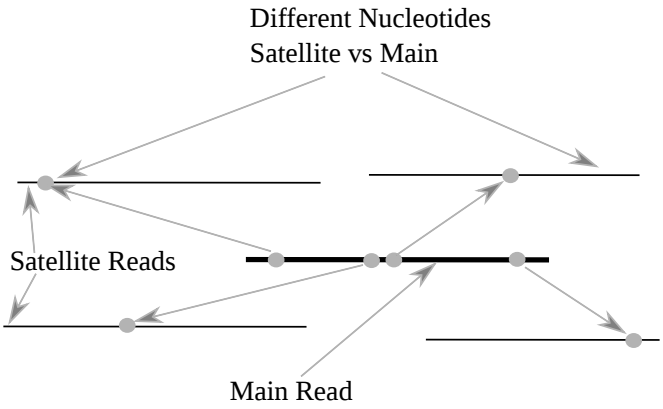
- two common inexact sub-k-mers with mismatches/indels on their prefixes or suffixes

Next, the program uses NW to align each read in a group against the seed, generating a partial alignment. These alignments are inserted in a POG, from where the corrector extracts the correct version of a read as a path between bases.

Read Cluster Based (rcb)

Read **C**lustering **B**ased (**rcb**) methods use different clustering methods to group reads which fit together. This group resembles the **msab** one, but the algorithms in it do not generate an alignment, but search for reads that are similar and choose a consensus which is the correct form for all these similar reads. Fig. 3.5 shows a central read (having the most common part with all the others) and its satellites. For simplicity, we only exemplify the one difference case. In our example all four satellites have one distinct nucleotide each. A corrector should group them together as they present a high degree of similarity, hence they are in fact clones of the same read, but with errors. Fig. 3.5 b) is an example (extracted from the bibliography[16]) where the consensus read is the one with the highest frequency. The other reads differ from the main read by just one nucleotide and also have much lower frequencies. Please keep in mind that the algorithms included here do not perform a multiple sequence alignment to determine the correct read, they just group them by differences and search for a valid consensus, the error-free existing read.

FreClu [16] uses an iterative approach to group reads. This clustering method, starts selecting the most likely error-free reads, and it cluster all the reads in a tree structure where each read on a lower level stochastically derives from its parent (which is more abundant). A difference of one nucleotide between the two



a)

Main Read	Reads	Frequency
	TACCCTGTAGATCCGAATTTGTG	9281
One mismatch sequences	T***C***CCCTGTAGATCCGAATTTGTG	59
	TACCCTGTAGATCCGAATTTG***A***G	1
	TACCCTGTAG***C***TCCGAATTTGTG	151
	TACCCTGTAGA***A***CCGAATTTGAG	5

b)

Figure 3.5: a) Clustering approach for one reference read and four related having one difference each; b) Real example with the main read market in bold and the satellites aligned and with the different locus market with bold and italic.

levels stays at base of the aforementioned parent-child relationship generating a structure the root of which is the most frequently observed sequence.

In [61], the authors describe an error modelling method based on a graph that connects the reads that have one and only one difference, first published in [132]. The authors extend the original approach to allow a pattern of error rates varying by base change pattern and position in the read, not working in a multiplicative fashion. The corrector first trims and filters the reads. Using the one nucleotide variance, the method assemble the graphs and creates a subset containing their largest disconnected sub-graphs, to deduce the error model and correct the graphs.

Probabilistic Models Based (pmb)

Probabilistic **M**odels **B**ased (**pmb**) methods use the Expectation Maximization (EM) algorithm to determine the correct base at each position by calculating the likelihood of the existing variants at that specific position. Basically, the problem of error correction boils down to selecting the right nucleotide at a certain position where two or more reads overlap and there is more than one choice. The pmb software base their approach on the fact that this problem has unknown parameters (unobserved component), in this case the correct base. As a result, using the existing input data (observed component) and maybe more information (like the error rate), they try to generate a model (after multiple iterations over the same data). This model can say with a certain degree of trustiness of the correction of a certain nucleotide. The EM alternates between two steps, the E (guessing the probability) and the M (re-estimating the model parameters using the new probability), until it converges to the desired model. Figure 3.6 presents the basic algorithm. For an extensive explanation of the EM algorithm, the reader should check the article of Do and Batzoglou from [133]. Different algorithms in

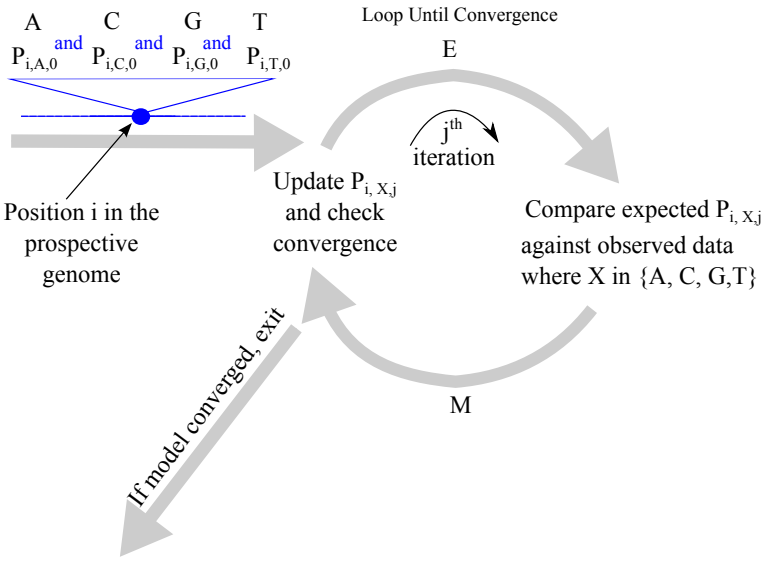


Figure 3.6: The EM algorithm initializes the probabilities of the bases before entering the loop where it alternates between E-step and M-step; Once the convergence threshold has been reached the method exits and enters the correction stage; The capital P represents the probability for a base to be the real one;

this category use different position comparison methods (position part of k-mer or read) and convergence points.

RECOUNT [62] uses the Expectation Maximization method [134, 135] to determine the true counts of a read based on the count of observed reads and the estimation of the error rate. A true read represents the real DNA sequence, whereas a observed read is the output of the sequencer. To calculate the observed count, the method creates a neighbourhood t for a seed read r including all the reads with a Hamming distance to r of one. The elements in t are the observed reads for the (considered as) true r.

Premier [136] models a sequencer as a Hidden Markov Model,

with each read being an independent realization of the HMM. It uses Expectation Maximization (EM) Baum-Welch to fit a HMM to a read. The authors view the problem of error correction from the point of view of a signal processing and error control. They present two algorithms for error correction: Viterbi and Fano.

Premier Turbo [137] expands upon the previous version from [136]. It allows errors in the first k-mer. Next, the new version applies the correction for a read in both the forward and reverse complementary directions. Lastly, the EM Baum-Welch method's (fitting the HMM's parameters) result at any stage is utilized for correction in the following stage.

kGEM [138] generates a haplotype set H for the input reads set R. Initially, H covers R with a number of mismatches. Haplotypes are turned into fractional haplotypes where each nucleotide can be one of the options: A, C, G, T or to be deleted. Running until convergence, (H maximizes PrR—H), **kGEM** determines the most frequent option at each position of each item in H. At the end, duplicates are collapsed and rare genotypes are removed generating the final version of H.

Please note that many methods can be included in more than one category. For instance, **Coral** [15] is listed in this review in the category of msb algorithms [1, 2, 79], but it also uses the k-mer spectrum to determine the related reads. The same case arises with **Premier**[136] and **Premier Turbo**[137] which use k-mers to update the probabilities for the variants on a position.

Recommendations

From table 3.1, it is clear that depending on the nature of the project some programs are better than others. For Illumina projects almost all correctors can be used. Although, as we shall see in the coming sections (mostly in section "Testing", the Illumina-only correctors offer a better performance on Illumina data when com-

Table 3.1: The software and important features support; The columns have the following meaning: Par. Tech- the parallel technology (if any), k- whether or not a software make use of k-mers, Q.- support for quality scores, N- support for uncalled bases, Indel- support for indels, V.L.- support for variable length reads, H.- support for heterozygosity, Rep.- support for repetitive regions, T.- whether or not the algorithm incorporates trimming, T.S. - the categories in which a software fits from Fig. 3.1

Type	Name	Par. Tech	Lang.	k	Q.	Tech	N	Indel	V.L.	H.	Rep.	T.	T.S.
ksb	**CUDA-EC**[65]	CUDA	c++	y	n	Illum.	NA	NA	NA	n	n	y	a
ksb	**Reptile**[98]	OMP	Perl/c++	y	y	Illum.	→A	n	NA	n	n	n	a
ksb	**Quake**[9]	OMP	c++	y	y	Illum.	NA	n	NA	y	y	y	a
ksb	**Edar**[99]	NA	NA	y	n	All	NA	y	y	n	y	n	a
ksb	**Hammer**[101]	-	c++	y	y	Illum.	y	n	y	n	n	n	a
ksb	**REDEEM**[102]	-	c++	y	n	Illum.	n	n	y	n	y	n	a
ksb	**DecGPU**[103]	CUDA/MPI	c++	y	n	Illum.	NA	NA	NA	n	n	y	a
ksb	**CUDA-EC2**[104]	CUDA	NA	y	y	Illum.	NA	n	y	n	n	y	a
ksb	**Qamar**[54]	PThr.	c++	y	n	Sanger/Illum.	y	n	y	n	n	n	d
ksb	**Parallel Reptile**[105]	MPI	c++	y	y	Illum.	→A	n	NA	n	n	n	a
ksb	**BayesHammer**[106]	OMP	c++, python	y	y	Illum.	n	n	NA	n	y	y	a

Table 3.1 Continued from previous page

Type	Name	Par. Tech	Lang.	k	Q.	Tech	N	Indel	V.L.	H.	Rep.	T.	T.S.
ksb	**QuorUM**[91]	PThr.	c++	y	y	Illum.	NA	n	y	n	n	y	a
ksb	**RACER**[107]	OMP	c++	y	n	Illum.	→A	n	NA	NA	n	n	a
ksb	**Musket**[13]	PThr./OMP	c++	y	n	Illum.	y	n	NA	n	y	n	a
ksb	**Hector**[52]	OMP	c++	y	n	454	y	y	y	n	n	n	b
ksb	**Lighter**[108]	PThr.	c++	y	y	Illum.	y	n	y	n	n	n	a
ksb	**HErCoOl**[110]	Threads	java	y	n	454/Ion	NA	y	y	n	n	n	d
ksb	**Trowel**[111]	Boost	c++	y	y	Illum.	NA	n	NA	n	y	n	a
ksb	**LoRDEC**[71]	PThr.	c++	y	n	PacBio	NA	y	y	n	y	y	a
ksb	**BLESS**[109]	-	c++	y	n	Illum.	n	n	n	n	n	n	a
ksb	**Blue**[10]	Threads	c#	y	y	Illum./454	y	y	y	n	y	n	c,e
ksb	**BFC**[112]	PThr.	C	y	y	Illum.	NA	n	y	y	n	n	a
ksb	**Scribble**[113]	NA	NA	y	n	Illum.	n	n	y	n	y	n	a
ksb	**PAGANtec**[93]	OpenMP	C++	y	n	Illum.	n	n	y	n	n	n	a
ksb	**ACE**[114]	OpenMP	C++	y	n	Illum.	y	n	n	y	n	n	a
ksb	**FADE**[115]	FPGA	Verilog	y	n	Illum.	y	n	n	n	n	n	a
ksb	**Pollux**[117]	-	C	y	n	Illum./Ion/454	y	y	y	n	n	n	d
ksb	Gu *et al.*[118]	NA	C++	y	n	Illum.	y	NA	NA	NA	NA	n	a
ksb	**Jabba**[120]	NA	C++	y	n	PacBio	n	y	y	y	n	y	c
stab	**SHREC**[64]	Threads	java	n	n	Illum.	n	n	n	n	n	n	a
stab	**HSHREC**[121]	Threads	java	n	n	All	y	y	y	y	n	n	d
stab	**PSAEC**[122]	PThr.	c/c++	n	n	Illum.	n	n	n	n	n	n	a
stab	**HiTEC**[6]	-	c/c++	n	n	All Subst.	n	n	n	n	y	n	a

Table 3.1 Continued from previous page

Type	Name	Par. Tech	Lang.	k	Q.	Tech	N	Indel	V.L.	H.	Rep.	T.	T.S.
stab	**MyHybrid**[12]	NA	NA	n		All	y	y	y	n	y	n	d
stab	**Pluribus**[124]	NA	NA	n	n	All	n	y	y	n	n	n	d
stab	**Fiona**[50]	OMP	c++	y	n	Illum./454/Ion	y	y	y	n	y	n	e
msab	**ECHO**[11]	Python/ PThread	Python/c++	y	y	Illum.	y	y	y	n	y	n	a
msab	**Coral**[15]	OMP	c	y	y	Illum./454	y	y	y	n	n	n	f
msab	**LSC**[73]	NA	Python	n	b	PacBios	y	y	y	n	n	y	c
msab	**CloudRS**[125]	Hadoop	Java	y	y	Illum.	NA	n	NA	NA	y	n	a
msab	Chung *et al.*[127]	Hadoop	Java	y	y	Illum.	n	n	n	NA	y	n	a
msab	**proovread**[72]	Grid	Perl	n	y	PacBio	NA	y	y	NA	y	y	c
msab	**Nanocorr**[77]	Grid	Python	n	n	Nanopore	y	y	y	n	n	n	c
msab	**Karect**[131]	PThr.	C++	y	y	Illum./454/Ion	y	y	y	y	y	y	d
rcb	**FreClu**[16]	-	java	n	n	Illum.	NA	n	n	n	n	n	a
rcb	Sleep *et al.*[61]	-	NA	n	n	Illum.	n	n	y	n	n	y	a
pmb	**RECOUNT**[62]	NA	c++	n	y	Illum.	NA	NA	NA	n	n	n	a
pmb	**Premier**[136]	NA	NA	y	y	Illum.	n	n	NA	n	n	n	a
pmb	**Premier Turbo**[137]	NA	NA	y	y	Illum.	n	n	NA	n	n	n	a
pmb	**kGEM**[138]	NA	NA	n	NA	All	y	y	y	n	n	n	d

pared to multi-technology software.

From a computational resources point of view, the correctors written in a low-level language like C++ should be used. One must take this last advice with a grain of salt as the performance is highly dependent on the quality of the code and the algorithms used. Another very important aspect is the multi-core and multi-computer support. Nowadays, even the mobile phones are multi-core and the speed of CPUs has hit a hard limit, therefore any piece of software capable of scaling on multiple cores should be preferred over the others. This scalable applications are very useful when the time frame is very short. Furthermore, the same programs win when testing multiple combinations of parameters at the same time and running multiple instances of a single threaded program is not an option (e.g. when the user has to run the next instance of the program with a combination of parameters based on a previous run). Lastly, the multi-threaded programs would normally consume less memory than multiple single-threaded instances running at the same time. The best example for this last observation is an OpenMP corrector creating a k-mer spectrum. An optimized multi-threaded program would create just one structure to keep the k-mers and their count and it would allow thread-safe access to the structure. An optimized single-threaded program using the same mechanism would avoid the locks but for multiple instances, the same structure would be replicated as many times as instances are running.

Discussion

In this section of the paper, we discuss the most important topics for the error correction state of the art. It starts with the general problem of errors in NGS data, followed by the key features of the methods and ending with the main testing approaches.

Challenges

Data Preparation and Post-processing Steps

There are cases in which the input data must pass through some additional pre-processing steps like the conversion to a certain format. **Blue** performs a preparation step to generate the k-mer spectrum. **Reptile** has a pre-processing step, to separate the reads from their quality scores and to filter reads containing ambiguous characters. **CloudRS** converts the FASTQ input file to a specific format. Furthermore, it must upload the converted input to the Hadoop cluster and download the result locally when the job has finished. **HSHREC** generates the corrected output files without the initial descriptions of the reads. Some tools may need this information for further processing like SolexaQA++ [192] which generates statistics from multiple technologies. The dataset requires a post-processing step (that the user must implement) to restore the initial information. Secondly, **HSHREC** generate two files, one containing all corrected reads and the other the skipped reads. Generally, the output of the error correction tools is FASTA/FASTQ and it does not require any explicit processing.

K-mer

K-mers Handling: Many methods base their decision on k-mers and apply different techniques to deal with the memory limitation and CPU requirements. **BLESS** uses the hard-drive to store the k-mers during the counting.[193, 194] **RACER** encodes the bases in a k-mer as a 2-bit representation to save memory. A newer version of **Quake** integrates Jellyfish [195] to count k-mers instead of its own implementation to stay competitive against the more recent algorithms. It also provides a distributed approach for those cases in which the local memory is not enough to handle the k-mers. To speed up the k-mer spectrum generation, [105] implements a par-

allel counter. [103] use GPGPUs to generate the k-mer spectrum. A rare feature is the support for variable k-mers for grouping reads as in [125], where the corrector uses a wildcard based k-mer.

K-mer Size Selection: The value of k is extremely important.[9, 136] A too low value for k would result in many k-mers appearing in most reads, thus joining in groups reads without any real relation. On the contrary, very large values would generate too many unique k-mers, which also have a higher probability of including more errors, therefore introducing noise into the grouping. Long k-mers may also require more RAM memory.

The analysed methods set the size of the k-mer by: accepting a user value, having a fixed default value and/or performing automatic selection. **Hammer** requires the user to set it. **Quake** and **ECHO** define a formula to determine the optimum value as presented in Table 3.2. **Reptile** considers $10 \leq k \leq 16$ enough for microbial genomes, fitting the spectrum in less than 4 GB of RAM. **Coral** uses a default value of 21 for the k-mer. **Hector** and **Musket** require both the k-mer length and the total estimated number of k-mers from all reads. **HiTEC** and **Fiona** automatically identify the optimum k-mer length at each step. **CloudRS** stacks reads using a wildcard based 25-mer and, later in the correction procedure, a fixed 24-mer. The k-mer size should be odd in order to avoid palindromic k-mers.[196] The software using k-mers in their pipeline are market in table 3.1, column "k".

K-mer Distribution: Software relying on correct and erroneous k-mers tries to fit the k-mer spectrum on a certain distribution. The correctors compute the histogram with the frequencies for each k-mer in the set of reads. A valid estimation tries to model the initial, complex distribution as a combination of multiple, simpler distributions.[9] **Quake** divides the solid k-mer distribution in a combination of a normal and a Zeta distribution and it considers (like **BLESS**) the weak k-mers to follow a Gamma distribution. **Lighter** assumes a Poisson distribution, like **Fiona**.

Table 3.2: Formulas to determine the k-mer size for non-automatic k-mer determination; N = Genome length, l = read length; p = probability that a random k-mer appears in a random string of length N, using the alphabet {A, C, G, T}; N_s = number of unique solid k-mers as reported by BLESS

Formula	Where
$k = \log_4 200N$	[9, 12]
$k \geq \lceil \log_4 N \rceil$	[15]
$4^k > N$	[98]
$k = \lfloor l/6 \rfloor$	[11]
$k = \log_4 2Np^{-1}$; $p = 10^{-4}$	[110]
$N_s/4^k \leq 0.0001$	[109]

REDEEM models the k-mer distribution as a Multinomial one. For 454, **Hector** encodes homopolymers using the base and the multiplicity. The authors observe that the distribution of the original reads tends to be unimodal. With the encoding applied in the homopolymers space, the distribution of the homopolymers spectrum is analogous to the one (bimodal) obtained by **Musket** in base space. The same authors conclude that, generally, the homopolymers spectra are bimodal.

Coverage Cut-off and K-mer Distance: K-mer based error correction methods can cut the k-mer histogram to remove k-mers with too high or too low frequencies, which normally reduces the noise caused by highly-repetitive regions or singular errors. Some algorithms automatically compute the best cut-off value, allowing users to override this value. **Quake** uses the Broyden-Fletcher-Goldfarb-Shanno method to calculate the histogram cut-off, but this method fails when the curve of the distribution is not smooth enough.[13] The authors of **Musket** empirically determined that the lowest count for a k-mer around the valley can be a good cut

point. **Musket** also has an option to use a user-provided value. **Trowel** uses a different approach by using the contiguity of high-quality-scores bases instead of the coverage. It also expands the trusted k-mer set, adding new k-mers after they are corrected. **QuorUM** bases its decision on the quality of bases from k-mers, therefore all bases in a solid k-mer must have a quality greater than a threshold. **RACER** uses an internal threshold to deem a k-mer followed or preceded by a certain base, either as solid or weak. **Hammer** and **Reptile** create a Hamming graph for the array of all k-mers, locating groups of similar k-mers that only differ in a few positions, and then collapsing all those k-mers into a consensus k-mer. To improve memory consumption, **Lighter** uses a random method to decide whether to store or not a k-mer, assuming that a correct k-mer appears multiple time in a dataset, thus the chance of being selected is high.

Bloom filter: The Bloom filter [197] requires less memory than a traditional hash-map, with the downside of occasional false-positives. We discovered that only the k-mer spectrum based methods employ it at the moment. These correctors use the structure to store the spectrum. **Musket** uses a Bloom filter in conjunction with a hash table to remove unique k-mers. Furthermore, it splits the work between multiple threads and instead of using a global Bloom filter, it assigns a local one to each thread to speed-up the processing. **BLESS** heavily relies on Bloom filters to perform correction.

Repetitive Regions

In general, the problem of repetitive regions cannot be tackled by considering individual reads or k-mers in isolation. Some argue that in the case of highly repetitive genomes, a sequencing error has a greater probability to change a solid k-mer to another solid k-mer.[9] They calculated the percentage of all one base mutations

for a k-mer k that will convert k into a sequence which also exists in the genome. The results show a 2.25% for *E. coli* and 13.8% for human chromosome 1, with a 15-mer and a 18-mer respectively. Increasing the k-mer length up to 19 did not significantly change the result, dropping the percentage to 11.1% for the *H. sapiens*' first chromosome. The different percentages obtained for the two organisms result from the higher complexity of the human genome. **REDEEM** was specifically designed to handle repeats. The main problem with repetitive regions is the similarity of two sequences which are reside on different loci of the genome. A corrector may try to convert them to a consensus, hence destroying the existing valid zones. These wrongly fixed reads would prevent an assembler from correctly composing the real genome (or make it generate chimeric assemblies). It is a real problem for highly-repetitive genomes (plants).[1] Furthermore, misreads in repetitive regions can cause an abnormal high frequency of a k-mer [97] which could result in an erroneous classification as solid by some correctors. The methods based on multiple sequence alignment are more resilient to challenges posed by repeats, although they do not totally solve the problem.[11] The relationships between reads can tackle to some extent some small repetitive regions because of the higher length of the analysed strings compared to k-mers. Moreover, a read from a repetitive region has a higher probability to enter the right group provided it shows enough dissimilarity with other repetitive regions. **Fiona** implements a filter to remove suffixes with an unreasonable high frequency and supports tandem repeats. **Blue** addresses the problem of repeats by evaluating alternative fixes for a read (it works in the case of significant differences among reads). **proovread** takes into account the loci on the long reads where large blocks of short reads map, collecting many reads. In contrast, non-repetitive regions may not even participate in alignments, because of the uniqueness resulted from the Pacific Biosciences's high error rate. [125] makes use of a high frequency k-

mer filter to avoid stacking reads from repetitive regions. Column "*Rep*" from table 3.1 contains the support for repetitive regions in the correctors.

Ploidy

A corrector must distinguish between errors and variants. Since most error correctors where tested on bacterial genomes, the information on the behaviour of most tools are restricted to the haploid case. The support for heterozygosity is stated on column "*Hzy.*" on table 3.1. Authors of [61] apply a smoothing technique to avoid removing zones with only biological variants. They build a tree using the frequency of reads, and consider a true variant as a sequence appearing with a high enough frequency compared with the parent sequence. Their decision is based on the fact that the frequencies of a variant should be much higher than the ones from the sequencing errors. The authors of **ECHO** explain a modification to support a diploid genome with homozygous and heterozygous genotypes. Their approach is to consider a uniform distribution over all possible genotypes. They skip a correction if the estimated coverage is much greater than the expected coverage at an analysed locus.

Read Trimming and Splitting

To avoid the propagation of errors to the next steps, the correctors may eliminate bases from both ends of a read, which can be considered a complementary method to reduce errors.[121] Users should take great care in using trimming, because it can heavily influence the next steps like assembly, where the final result may become fragmented [91] as the assembler is not able to find proper overlaps among reads. However, not trimming faulty bases may result in erroneous assemblies.[78] Some authors [103, 96, 65] try to fix the read first and when this is not possible, they trim it. If

the result remains unsatisfactory, they discard the read. Another approach is to pre-process the reads and cleave the suffixes and prefixes having low quality scores to decrease the number of false positives in the k-mer spectrum.[104] One must be careful with the quality scores and take into account that the accuracy of the quality scores depend upon library preparation method as demonstrated for Illumina MiSeq data [57]. When dealing with very long reads like Pacific Biosciences with a high percentage of errors [71], along with considering trimming their ends, an additional approach is to split the long reads into smaller, high quality segments. The authors of [99] do not correct the spurious bases, opting instead to split the read at the locus of the error and to remove the faulty base. They argue that for a further assembly step based on k-mers such as Velvet [196], their method should not pose any problems.

Unknown/Uncalled Bases

The unknown bases (denoted by N in Illumina and Roche 454 sequencing) are one of the four basic types of errors. Schirmer *et al.* concludes that this type of error does not occur randomly as supported by their non-uniform distribution. Column "*N*" from table 3.1 indicate whether the correctors support Ns or not. In some cases [61, 64, 6, 121], the authors prefer to exclude all reads which contain one or more uncalled bases. Other programs (such as **LSC**) eliminate the reads that have a frequency of Ns above a threshold. A different strategy is to convert each unknown to a real base, like **RACER**, **Reptile** and **Parallel Reptile**. Tools such as [15, 10] tackle the unknown bases in the correction process. Some authors fail to mention the support for uncalled bases in their papers, therefore it is up to the user to experiment. Since the unknown bases are a type of error [50], the authors should make it clear whether their software supports them or not. In [10], the authors put together a table with a selection of correcting software

and their features including handling of uncalled bases.

Low-Coverage Regions and Uniformity

Illumina sequencing generally has a higher average coverage than other platforms [198], but their short size may not be suited for phylogenetic profiling when a high resolution is required. However, many studies [199, 200, 63, 201] have found that GC-poor and GC-rich regions have low coverage or even no coverage at all. An error corrector must consider these platform-specific shortcomings to increase sensitivity and specificity.[9]

The authors of **MyHybrid** and **Coral** state that correction methods expects the coverage to be relatively high and use that multiplicity for a meaningful decision. Therefore, they cannot do much for reads from low-coverage regions. The methods that use a threshold for weak/solid k-mers will work if the coverage is high enough or uniform, but they will end up destroying the low-coverage regions (**QuorUM** tries to avoid this problem). **Edar** takes into account the bias introduced by GC regions when calculating the k-mer coverage, by actively considering the GC content of the k-mer. For a reliable result, the authors recommend using a reference genome to accurately calculate the coverage. While many authors do not state the minimum required coverage for a successful correction, **ECHO**'s paper specifies a coverage of 15 or higher. **Hammer** and **BayesHammer** are specifically designed for error correction without uniformity assumptions. Due to uniformity, some authors admit their algorithm's limit like in case of **Reptile** where a non-uniform coverage and the existence of more than one acceptable tile force the algorithm to skip a correction decision. The authors of **Blue** mention the caveat of a simple k-mer cut-off due to uniformity which can result in the rejection of correct k-mers in low-coverage regions and the acceptance of erroneous k-mers in very high-coverage regions. **Fiona** detects erroneous k-

mers by calculating the expected coverage for each k-mer, given a uniform sampling of genomic positions. It uses a hierarchical statistical model to describe the expected coverage distribution of k-mers based on library preparation and sequencing.

Parameters

All the methods rely on specific thresholds, lengths, ratios or probabilities to drive their correction. As the methods evolve, they tend to move the burden of choosing the best suited values for their parameter from the user to the program itself.

Generally, the k-mer size has a default (user adjustable) value. However, some correctors like **Quake** require an explicit value from the user, but offer a formula to determine it. **Coral** has a default value, but it also proposes a formula in order to obtain the best results.

The technology flag can explicitly set the source technology. For example, **Coral** has a flag for "Illumina/454" and **Fiona** another one for "Illumina/Ion Torrent/454", which helps the software to decide the best approach for correction. Others have a flag which enable targeting the errors specific to a specific platform, i.e. homopolymers errors in 454 with the "hp" flag in [10]. **Karect** can run with indels support (Ion Torrent, Roche 454) or without (Illumina).

Parameter selection automatic/manual: A manual method to select parameters requires the user to try different values for different parameters to obtain the best results. On the other hand, an automatic method would prevent the user to provide additional valuable information to infer the best actions the algorithm has to take during the correction process. We must distinguish between automatic determination of the best value for the dataset/ analysed case and the default value of a parameter (deemed by the authors to be an acceptable value). The two programs supporting full

automatic parameter value selection are **HiTEC** and **Fiona**. Note that **HiTEC** needs the length of the genome and the percentage of errors as input, but these two parameters remain the same for a certain dataset.

Single Threaded vs Parallel

Generally, the programs tested in this review support parallel processing using multiple threads. There are methods, like **BLESS**, that can compete against multi-threaded software due to their approach, despite being single threaded . Other methods like **Reptile** have been updated to run on multiple CPUs [105], using the same initial correction mechanism. The parallel implementations are a normal trend as both the NGS data size and the length of the reads increase.

A distinction must be made between those programs being natively parallel (they internally split the jobs between multiple workers) like **Coral** and **HSHREC** and those that have no parallel implementation but their input can be divided in chunks and multiple processes can be launched on different fragments of the initial dataset like in the case of **proovread**.

Parallel Technology: The reader can check the parallel technology used by a corrector on column "*Par. Tech.*" from table 3.1. Most of the parallel implementations use OpenMP to distribute the workload among the threads. **DecGPU** and **Parallel Reptile** support distributed-memory computing by MPI. The distributed-memory model requires a more complex programming and configuration, but enables gathering a larger number of RAM memory. This advantage may enable tackling larger-scale correction problems which cannot be addressed on a single node. Other methods, such as **Quake**, use parallelism only for counting k-mers. Furthermore, **proovread** and **Nanocorr** can run on multi-core desktops and distributed clusters using queuing engines like SGE (commer-

cial[2]) and SLURM [202]. As of now, there are two fully distributed methods using the Map-Reduce (Hadoop) paradigm.[125, 127] An interesting addition to the field is **FADE**[115], the FPGA error corrector which unleashes the massive parallelism available on FPGA devices to tackle the error correction.

Operating System and Programming Language

The resource consumption is a problem because of the continuous growth of the size of the NGS data. Owing to this, we observe that the majority of authors chose a low-level language like C or C++ to implement their solution. An interesting trend is the use of of C++ over C in writing the software, with just one program (from those being available online for us to analyse), **Coral**, being implemented in pure C. The C# implementation of **Blue** obtains the best performance when compared against other algorithms on Illumina and Roche 454. Even though C# is not considered to be a high performance language, **Blue** performs really well against the rest of the algorithms. Some correctors appear to be implemented in Perl and/or Python. These are often scripts that are used to execute third party software. In case of **LCS**, the authors offer a software wrapper written in Perl that uses an external aligner to map the short reads against the long ones. **Nanocorr** is a python wrapper for BLAST and pbdagcon [3]. For the reader's convenience, we list in table 3.1 the programming language of choice for each corrector.

Overall, the correctors should work on the three most important desktop/server operating systems: Windows, Linux and Mac OS. Some authors [108] mention the supported platform in their papers. Generally, the authors prefer to support Unix flavours and to distribute the source code and the instructions to build it.

[2]Available at `http://www.univa.com/products/grid-engine.php`
[3]Available at `https://github.com/PacificBiosciences/pbdagcon`

Furthermore, the tests for the majority of the works were done under Linux, with some authors also using Windows.[61] Besides PC based methods, we included a corrector which runs on FPGAs - **FADE** - (even though with the help of a computer that handles the data transfer and storage).

License and availability:

The majority of the software is available online free of charge. As we specified in the inclusion criteria, we focused on freely available software appearing in a journal or conference. Some papers do not include a link to the software, and some of them do not even include any mention to its availability. Releasing the software online is a method to attract more users and to improve the existing solution using their feedback.

Recommendations

Depending upon the preprocessing type, a corrector that can separate the steps of the correction can save a lot of time when processing big datasets. For instance, **Blue** can generate the k-mer table in a separate process. In its case, the advantage is twofold. Firstly, it can use the same k-mer table for multiple runs with the different combinations of values for the majority of parameters that are not involved in the histogram's generation. Secondly, it makes cross-correction possible since the k-mer table can be generated from one technology that can be used to correct data from a different one. Another important aspect is the k-mer size selection. Any program able to determine the size of k-mers automatically (**HiTEC**) or use variable sizes (**Fiona**) is recommended over those with user-defined only k-mer size selection. Next recommendation is to select a software like **Fiona** or **Blue** that consider repetitive regions as they may avoid altering similar zones with SNPs.

The unknown bases support depends upon the used sequencing technology as some technologies do not produce this type of error. Furthermore, in case of extreme necessity, one can easily write a script that can convert the unknowns into random or specific nucleotides. The low-coverage issue must always be a top priority since some algorithms can skip those zones because of the limited information. Finally, the user must be careful with the trimming and the splitting of the reads. In the former case, a corrector (like **Quake** and **DecGPU**) may trim a read if the correction is not possible. If result is then fed to an assembler, the correction may negatively influence the overlap detection. **Edar** applies a distinct correction mechanism by cleaving the reads. This approach may be detrimental for any further step because a lot of information is lost. The reads become much smaller and the relation between segments part of the same read is lost forever.

Testing

This section focuses on the testing part included on the analysed papers. We extracted all datasets which we could identify in the papers along with the results provided by the authors. Due to space limitation, we split the datasets in two categories. Table B.1 lists all those datasets appearing in [1, 2]. The rest of the datasets (second category) along with the benchmark information are located in the Supplementary Material. The gain metric for both categories is calculated (by the authors or by the reviewers) using the formula from [1]. The reader must be careful though, because there is no standard way to count TP, FP and FN. As a result, the numbers given by the authors and the reviewers must be taken with a grain of salt. The reliable gain appears on the same column (same review, the authors used the same testing approach for all software). The hardware configurations used by the correctors'

authors and in [2] as more sections related to testing are located in the Supplementary Material.

Methods

Some metrics are general and do not focus on certain type of error correction mechanism. For instance, the sensitivity and specificity appear in k-mer-spectrum methods like **Reptile**, suffix trie/array methods like **SHREC** and MSA methods like **Coral**. Sensitivity, specificity, gain and genome assembly statistics are the most widespread metrics.

Simply counting the mapping reads that did not map before the correction and do so after it, can prove the effectiveness of a corrector.[62, 121, 73] However, the differences obtained heavily depend on the aligner's parameters. For example, the authors of [1] test with different values for the aligner, albeit only for datasets with indels, given the complexity introduced by these types of errors.

In the case of artificial datasets, it is possible to report quite reasonable the error rate before and after correction.[99, 11] On the other hand, the exact error rate for real data can only be estimated.[53]

Gain/Specificity/Sensitivity

The gain (G), specificity (SP) and sensitivity (SE) metrics (for formulae see Equation 3.1) seem to become the de-facto on error correction. SP and SE first appeared in [64]. The gain[98], represents the percentage of eliminated errors. They are all based on counting:

- TP (true positives) existing errors that are corrected.
- TN (true negatives) correct bases left unmodified.

- FP (false positives) correct bases that are wrongly considered being faulty.

- FN (false negatives) erroneous bases left unmodified.

$$G = \frac{TP - FP}{TP + FN}, SE = \frac{TP}{TP + FN}, SP = \frac{TN}{TN + FP}. \quad (3.1)$$

There are differences in how the authors of each tool compute TP/TN/FP/FN. For **Reptile**, they compute the errors at base level, while for **SHREC** and **RACER**, they count the errors at reads level (a read is either error-free or erroneous, without considering the number bad base). The lack of a standard approach on counting the errors leads to some serious inconsistencies in the results published in the literature by the same tool in different benchmarks even using the same dataset and formula. For example, in [1], **Coral** obtains a score of 0.002 for an Illumina dataset (SRR022918), while with the same dataset, it scores 0.97 in its own paper. There is no doubt that **Coral** is a good corrector (as demonstrated by [50]) and even in the aforementioned survey it performs really good on datasets with indels. The problem lies in the different approach in performing the tests, which is not infallible. Moreover, the previous difference in score may arise just by changing the way to prepare the datasets before correction. Even though the approach is the same (filter non and multi mapping reads), the aligner can make a difference too. Salmela *et al.*[15] use Soap, while Yang *et al.*[1] use BWA.[203, 204] Table B.1 contains the results obtained by some correctors in [1, 2] on a number of datasets. The different results obtained in the original article versus the surveys can be explained by the difference in dataset preparation, FP/TP/FN/TN counting and maybe different versions of the tested programs.

Yang *et al.* and Tahir *et al.* have tested a number of algorithms with real datasets from different technologies. Both papers report the quality of correction (TN/FN/FP/TP, Sensitivity, Specificity and Gain). They even use the same formulae to calculate the Sensitivity, the Specificity and the Gain.

After reading the previous paragraph, the vigilant reader may asked him/herself why the numbers are not the same (one dataset is common for both surveys). The results differ because, even though the general idea is the same, their approaches are slightly different. Yang *et al.* map the original, uncorrected datasets against the reference genomes of the organisms that each dataset was obtained from. They use BWA[]/RMAP[] to map the original Illumina datasets, Mosaik[] for the original 454 dataset and TMAP[] for the original Ion Torrent dataset. Using the mapping results, they select only the reads that map exactly once. The differences between each read and the reference genome are stored in E_m. Next, two cases arise:

- for mismatches only type of errors, the Hamming distance between the original read and its corrected counterpart is calculated and the differences are stored in E_c; The following formulae are used to determine FN/FP/TP:
  - TP $= |E_c \cap E_m|$
  - FP $= |E_c \setminus E_m|$
  - FN $= |E_m \setminus E_c|$
- for indels, the corrected read is aligned (global alignment) against the the genomic region where the mapper determined that the original uncorrected, read fits; The differences in the global alignment are stored in E_r and FN/FP/TP result from the following formulae:
  - TP $= |E_m \setminus E_r|$

- FP $= |E_r \setminus E_m|$
- FN $= |E_r \cap E_c|$

Finally, the authors use the formulae from the main manuscript to calculate the Sensitivity, the Specificity and the Gain Lastly, Yang *et al.* also report the memory consumption and the running time for each dataset.

Assembly

The majority of the recent publications include some information about the assembly performance. Many list the N50 metric and the contigs count, but there are variations. Salzberg *et al.*[78] define N50 value as "the size of the smallest contig (or scaffold) such that 50% of the genome is contained in contigs of size N50 or larger". A contig is a multiple sequence alignment of reads represented as a consensus while the scaffold is a list of contigs that defines their order, orientation and the length of the gaps between them.[205] **Pluribus**' paper provides the number of nodes in the Bruijn graph generated by Velvet, which give a measure of the fragmentation of the assembly. For **QuorUM**, the E-size statistics [78] complements the N50 value.

While N50 and the maximum contig length measure the quality of error correction and give valuable feedback over the correction, the authors of[10] state that these metrics are not always accurate. This mainly happens because the assemblers can generate chimeric contigs in overcorrected datasets. In any case, the correctness of an assembly is hard to verify.[206] As a consequence, more refined assembly evaluation approaches are considered (e.g. the Mauve Assembly Metrics [207] for **Blue**). **BLESS** uses several assemblers from two different categories (de Bruijn and string-graph based). This approach increases the reliability of the capability to produce valid results that do not fit a certain type of assembler (or worse,

a certain assembler) and to prove the corrector's generality. Furthermore, the same authors do not just provide the value of N50, they also assess the quality of the assembly, using GAGE [78], as the authors of **Musket**.

For **BLESS**, the authors chose only artificial datasets to demonstrate the capabilities of their implementation. **DecGPU** contains the assembly information only for their corrector, not the other correctors in their benchmark. Salmela[15] eliminates the trimmed reads generated by **Quake**, because the Illumina-only assembler Edena[208] can only handle reads with the same length.

Our reader can find the assembly results (where available in the original paper) in the table with the performance assessment from the Supplementary Material.

Genomes Used for Testing

A recent review [79] tests seven correctors on three large genomes (*H. sapiens*, *D. melanogaster* and *C. elegans*) among others. The datasets for the aforementioned species are very large, between 31 million and 1.7 billion reads. As discussed in [11], the more complex, diploid genomes bring up the problem of heterozygosity and how to discriminate between true variants and sequencing errors. The repetitive regions also pose a problem to the corrector as mentioned by the authors of **Musket** and **HiTEC**. We can see a clear focus on the human genome, as the largest and most complex datasets for benchmarking come from this organism (for **BLESS**, **Blue** and **Fiona**).

Real vs. Artificial Datasets

We discern three situations for the datasets used in benchmarks: correctors tested only on simulated datasets, only on real data or both types of datasets. The main reasons stated to avoid artificial

datasets are the lack of simulators capable of producing meaningful data and the non-existence of some real challenges that only appear in already existing real data [15, 98]. To generate the artificial data, the authors of **BLESS** used simLibrary and simNGS [209], for **Lighter** and **Hector** - Mason [210] and for **Pluribus** - ART[211]. Pbsim [130] is cited in [72], but the authors did not use it, as they preferred to perform their tests on real data only. Correction software from 2013 onwards are tested mainly on data generated with dedicated software, opposed to previous use of in-house mechanisms. Some authors test their work on existing artificial datasets, e.g. [97] for **Edar** and [212] for **Qamar**.

Resource Consumption

Despite that early methods were not explicitly targeting a reduction in the requirements on CPU and memory consumption, currently all the methods try to address this aspect. Some of the first stand alone error correction methods were developed in Java, whilst the latest prefer C/C++. On the other side, **Blue** (2014) runs on the Microsoft .Net® platform while offering a very good performance. Many authors do not specify the exact method of determining the resource consumption. The authors of [71, 15] use the Unix time command.

There are multiple approaches to measure memory consumption. This is especially troublesome when testing programs in C/C++ against the ones in Java. For the latter, the simplest way is to measure the memory used by a process, but one must be careful with the memory allocation of the VM. In case of very small datasets, the overhead added by the VM can give a skewed view of the real behaviour. Even the memory usage of native programs is hard to assess under Linux given the fact that there are two main memory types: virtual memory and Resident Set Size. Most of the papers do not state how memory is measured, mak-

ing comparisons difficult. We also observed a lack of information regarding the number of threads used to asses the performance in some cases. For example, [64] state the use of multi-cores, but do to mention how many threads were actually used in their test. The scalability of the program, i.e. how the software behaves with an increasing number of threads is often not evaluated. Some authors like [10] present additional test cases in which they only asses their program, usually for very large datasets. Table B.1 contains the results for a number of correctors. For reader's convenience, we also included the results obtained in the original articles. One can see some differences in the memory and time obtained on the same dataset. This is not necessarily a sign of overinflated results in the original work, but more of a difference in the testing system and how the authors prepared the data for correction. Furthermore, there is no clear indication of the version of the program used in the original benchmark and survey benchmark respectively. The aforementioned table is a guideline for the interested user that can help him/her choose the right toll given the target hardware.

Testing details

This section presents all the methods and information regarding the testing methods in Table C.1. This should come in handy for any developer who may be curious how the already published correctors were tested and against what genomes.

Next, the actual results obtained by the algorithms, ordered by dataset are listed in Table D.1. It also contains the N50 assembly score to offer a second perspective of the correction performance.

Finally, the section ends with a list of the hardware equipment used by the authors in Table D.2. This information is valuable especially in the case of the Runtime where one can get a pretty good idea at what he/she can expect from a certain corrector.

Recommendations

Using the three benchmark reviews cited earlier, we can see some correctors that emerge as the winners. Molnar and Ilie[79] consider **BLESS**, **Musket**, **RACER** and **SGA** the best choices for HiSeq data. The last three were also able to handle H. Sapiens datasets with over 1.3 billion reads of 100-103bp on a Dell computer with 32 cores and 1TB RAM. For MiSeq, **RACER** wins in three from a total of four tests. Tahir *et al.*[2] recommend **HiTEC**, **ECHO** and **DecGPU**. In their opinion, the first two have a plus with their automatic parameter selection that can optimize performance. Finally, Yang *et al.*[1] obtain good results with **Reptile** and **HiTEC** for Illumina data. Unfortunately, **HiTEC** fails to run for three of seven datasets. This review also includes datasets from Roche 454 and Ion Torrent, where **Coral** wins in all cases against **HSHREC**. The two aforementioned correctors supporting indels do not obtain a high gain for Illumina datasets, making them inferior to Illumina-only correctors.

An important advice for the reader is to consider more than one metric when selecting a program. We recommend that (s)he should consider not only gain, sensitivity and specificity, but also other metrics like genome assembly and short read alignment. Another important aspect is the type of datasets used in testing. A corrector able to handle heterozygous organisms such as H. Sapiens (like **Blue**, **Fiona** and **BFC**) should perform pretty well with other complex organisms. From a resource consumption perspective, **BLESS** uses the least memory, but it is single threaded and quite slow since it uses the disk. **Blue** on the other hand obtains some very good results in its own publication, offering a trade-off between memory and CPU consumption.

CHAPTER 4

MuffinEC - Error Corrector

Part of this chapter has been published as: **Alic AS.**, Tomas A., Medina I., Blanquer I.. MuffinEc: Error correction for de Novo assembly via greedy partitioning and sequence alignment. *In:* Information Sciences. 2016. 329:206-19. DOI: 10.1016/j.ins.2015.09.012

An earlier version of the algorithm has been presented as: **Alic AS.**, Tomas A., Salavert J., Medina I., Blanquer I.. Robust Error Correction for De Novo Assembly via Spectral Partitioning and Sequence Alignment. *At:* 2nd International Work-Conference on Bioinformatics and Biomedical Engineering (IWBBIO). 2014. 1040-1048

Current NGS (Next Generation Sequencing) technologies offer the possibility of sequencing (very) large genomes and even whole populations (like thousands of individuals). However, errors introduced by sequencing techniques can still significantly affect the outcome of different ulterior processes applied to raw data like assembly [219, 220, 196]. Therefore, an error correction step can have a huge impact on the quality of de Novo assembly.

Most de Novo (without a reference genome) error correction methods can only detect and repair base mismatches. Despite this being adequate for Illumina reads, in which the predominant errors generated are mismatches [56, 221], other technologies are prone to generating indels [7]. In this paper, we focus on error correction methods that are stand-alone solutions (not included in an assembler package) and that are able to correct insertions and deletions.

We introduce MuffinEC, a corrector that combines a kmer approach for greedy read grouping with the Smith-Waterman (SW) algorithm for error detection and correction. Our method handles indels, mismatches and uncalled bases from variable length reads. The main objective of MuffinEC is to support different sequencing technologies while minimizing resource consumption.

Although indel-aware solutions are capable of working with basically any type of input data, just supporting indels is not sufficient and some parameters require adjustments for optimal results. For example, HSHREC does not have any tweaks for specific platforms, and its results are not on par with other solutions as shown by [1].

MuffinEC targets four platforms, namely Illumina, Roche 454, Ion Torrent and PacBio. However, as it accepts any type of error, MuffinEC can work with basically any existing technology, even though fine tuning of the parameters may be required to obtain satisfactory results. With the right technology profile, MuffinEC can be adjusted to fit different technologies.

A limitation found in some methods targeting just Illumina reads (which generally have the same length) is a fixed read length [6, 122]. For the rest of the platforms the read sizes vary greatly, thus it is very important for a multiple technology aware error correction software to allow variable length.

Sequencing technologies may not be able to interpret a certain nucleotide, marking it as unknown or uncalled. Some methods [64] ignore those reads containing unknown nucleotides. Others [107] choose to randomly change them to one of the four DNA bases (adenine, guanine, thymine, cytosine). This approach allows increased performance because a nucleotide can be represented with only 2 bits, but it loses accuracy. MuffinEC encodes the nucleotides in more than 2 bits to obtain the most accurate results when correcting a group of reads.

Another shortcoming of the available methods is their high computational resource demands. As the growth rate of the NGS sequencing data is faster than Moore's law [222], a problem arises because one may not be able to process new data without a costly hardware upgrade. The design of MuffinEC mainly targets memory consumption. MuffinEC also supports multithreading to fully exploit the potential of the multicore processors.

This chapter is divided as follows: the first section presents all the steps of the method, and the second section gives some relevant implementation details. The third section presents experimental results with the testing methodology most employed in the literature, and it compares the performance of MuffinEC against Coral, HSHREC, Fiona and HECTOR. The last section presents the paper's conclusions and outlines future research directions.

Materials and Methods

To preserve consistency across the whole chapter, we define: G as the prospective genome from which the reads originate, R as the initial set of reads, r_i as the ith read in R, *KM(r)* as the set of all k-mers for a read, *KM+* as the set of all k-mers for all elements in R, *M(KM+, R)* as the map having the keys as k-mers and the values the ids of those reads containing the kmer once or more times, $N(r_{seed})$ as a group of reads having a certain number of k-mers in common with r_{seed}, B as the set of all bases A, C, G, T, N, - (N = unknown base, - = missing base used when marking an insert)

The set KM is obtained by generating all k-mers for a read. MuffinEC obtains KM using a sliding window of size k and moving it one nucleotide at a time. At each step, a new kmer km is generated. When the sliding window encounters an unknown base, it discards the whole content and searches for the next k real bases. By discarding all k-mers containing unknowns, we can employ a 2-bit representation for the elements in *KM+* and KM. Note that we only use the 2-bit representation for the kmer table. This is highly desirable because the kmer set consumes a large amount of memory. A 64-bit integer can hold up to 32 bases, thus the maximum size of km is 32. The set *KM+* is obtained by the reunion of all KM for each read. MuffinEC corrects the groups of reads using the full 1-byte per base (ASCII) representation of a nucleotide to allow uncalled bases.

Owing to the built-in FASTQ support, MuffinEC is able to trim erroneous bases to improve the quality of the output. Furthermore, the quality of the generated k-mers can be substantially increased by eliminating low quality bases from the both ends.

Figure 4.1 presents the main working mechanism of our program, based on master-slave threading model

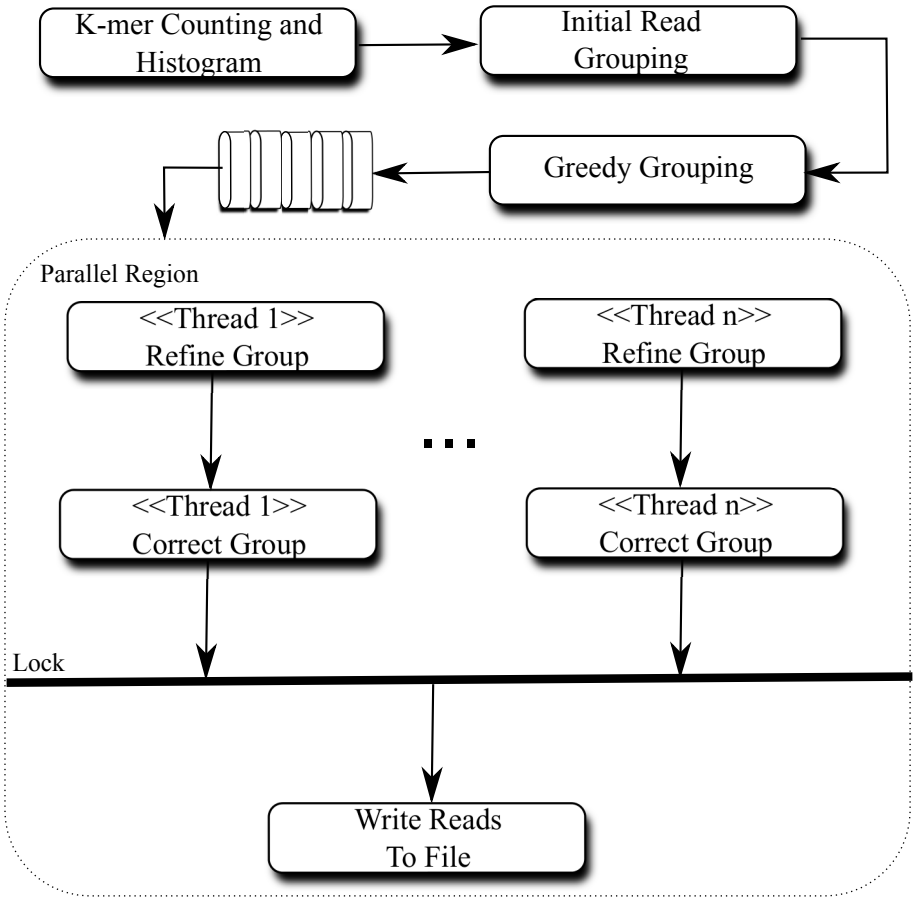


Figure 4.1: The basic flow of the algorithm.

MuffinEC has a total of five main steps, summarized below:

- k-mers counting and k-mers histogram creation with the purpose of eliminating low frequency k-mers,
- generate single k-mer relationships between reads,
- greedy grouping of the reads based on the number of common

k-mers as a distance metric

- refinement of the groups using SW as a distance metric to detect those reads that are truly close; at this step the original group may be split into subgroups
- correction of the subgroups (or the original group if no split was necessary)

The next five subsections detail each of these steps.

k-mers Count and Histogram

The first step in MuffinEC is the k-mer counting. It generates *KM+* along with the k-mer multiplicity for each $km \in KM+$. We eliminate all k-mers that appear less than the minimum size of a group accepted for correction (default minimum value being 3). In contrast to other k-mer based methods that correct k-mers by making use of *KM+*, we do not modify the eliminated k-mers. We merely use them to remove many false relationships between reads, as many of these low count k-mers are erroneous [9]. In addition to increasing the quality of the groups, this step also improves memory consumption.

One of the reasons for avoiding the k-mer histogram correction only is because its vulnerability to low coverage regions. The coverage is an important aspect when using NGS data. Due to its multi-step grouping mechanism, MuffinEC is sensible to coverage only when it is creating the initial groups. Let's consider five overlapping reads *A*, *B*, *C*, *D* and *E*. We have the minimum group size set to three (default value and minimum size for a group to make a correction possible). MuffinEC ignores the reads in groups with a size less than the aforementioned minimum. If we have the case in which *A*, *B* and *C* overlap and *C*, *D* and *E* overlap too, because of the greedy approach only group *A*, *B*, *C* is created. *D* and *E* form

their own group, but its size is smaller than three. This limitation only appears in border cases when the coverage is very low. In this border case, even the correction method may have problems, because the groups are so small that the correct base cannot be derived reliably. For the same case, a k-mer-histogram approach may entirely eliminate the whole low coverage area because the count of the k-mers is very low (and k-mers shorter). The MSA methods (like MuffinEC) have a tremendous advantage over the k-mer-histogram methods, because they have access to a lot more information given by whole, overlapping reads (which are longer than k-mers alone).

Initial Reads Grouping

The second step groups the reads based on their common k-mers. MuffinEC generates a new *KM+* (using the old *KM+*) to determine which reads can be clustered. Instead of merely counting, *KM+* uses the k-mers as keys and the sets of reads containing a k-mer key as values. We skip all those k-mers previously removed. This step creates basic groups and reduces the number of costly comparisons we have to perform at later stages.

Greedy Grouping

MuffinEC utilizes a greedy approach, i.e. it puts a read in the first group that satisfies the number of common k-mers condition. This approach (once processed, reads are discarded) is faster than the one in Coral (presented in chapter 2 - multiple corrections, same read) and it also saves memory (it produces smaller groups). As shown in the results section, it also increases the percentage of corrected errors. To avoid spurious large groups with unrelated reads, our software demands a maximum size of a group. When this requirement fails, MuffinEC selects a number of reads equal to the

group's maximum size (which is user adjustable). This step is not dangerous at all for very high coverage (true) regions because it merely splits the large groups into multiple smaller ones. Furthermore, the maximum group size is generally high enough to provide sufficient coverage for efficient ulterior correction. At this step, we reverse complement a read before we correct it at the next step if it fits in a certain group. The group creation is in essence an exploration of a weighted undirected graph *GR=(R, E)* (an example in Figure 4.2). E contains all these connections between the reads.

The algorithm starts by setting the first read from the input file as r_{seed}. It creates *KM*(r_{seed}) and traverses G using *M(KM+, R)* to search for reads having at least one k-mer in common (denoted by $c_{k\text{-}mers}$ or k-mers count) with r_{seed}. When searching the graph, we also convert each $km \in KM$ to its reverse complement, km_{rc}, and search for those reads containing km_{rc} too. Depending upon the direction of the common k-mers, MuffinEC changes the direction of the read added to the group.

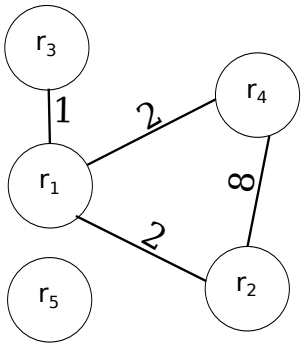
After the related reads are found, MuffinEC verifies each read if it has at least $t_{k\text{-}mers}$ (or a k-mer threshold) in common with r_{seed}. MuffinEC calculates $t_{k\text{-}mers}$ on the fly, for each compared pair of reads. To calculate $t_{k\text{-}mers}$, it uses the percentage of overlap between two reads, $p_{overlap}$. We allow a user adjustable percentage value instead of a fixed value to avoid fail cases where there is a very long read compared against a very short read. To better model real life situations, we also include the estimated percentage of errors p_{errors} in the overlap. We calculate $t_{k\text{-}mers}$ using the shortest read of the two compared using the formulae (where o_{size} is the size of the overlap):

$o_{size} = p_{k\text{-}mers} * min(len(r_{seed}), len(r_i))$

$t_{k\text{-}mers} = o_{size} - o_{size} * p_{errors} * len(km)$

Figure 4.2 show an example of a set of reads and their associated graph.

To keep resource demands low, our algorithm allows k-mer over-



Kmer	r1	r2	r3	r4	r5
AAGTAT	y	y	n	y	n
CCTCCT	y	n	y	n	n
CATCGA	y	y	n	y	n
AGTATC	n	y	n	y	n
GTATCA	n	y	n	y	n
TATCAA	n	y	n	y	n
ATCAAC	n	y	n	y	n
TCAACA	n	y	n	y	n
CAACAT	n	y	n	y	n

r1: GT***AAGTAT***GCCTCCTA**CATCGA**TGT
r2: ***AAGTAT****CAACAT*N**CATCGA**
r3: AACCTCCTNNCANNATNCACCGAT
r4: GC***AAGTAT****CAACAT*T**CATCGA**
r5: CGAANNNNCATTTACCCCCGAA

Figure 4.2: Graph example for five reads and k=6; The number of common k-mers gives the weight of a link between two reads; A missing link means zero k-mers in common;

lapping in contrast to other correction mechanisms that require non-overlapping k-mers [11]. This approach reduces the memory footprint (no k-mer location needed) and the overall execution time (no need to check if a k-mer is unique and if it is in a zone where no previous k-mer appears). Furthermore, to reduce memory consumption and increase execution speed, we also allow the common k-mers to appear in any order. Once the group containing r_{seed} is complete, MuffinEC sends it for correction to a worker (thread). The grouping process using k-mers continues for the rest of the reads that are still left unprocessed. Owing to this grouping method, MuffinEC is able to run fast, but it requires fairly high coverage to enable it to correct the groups.

Table 4.1: Consensus Example (the first five rows show the alignment example; the rest of the table shows the actual distribution values stored in the consensus)

	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

Group Refining

Once the previous step has generated a group of reads N, we use a modified version of the SW algorithm to refine it. SW is a well-known dynamic programming algorithm that computes a matrix of scores between two strings. The matrix contains all the possible alignments between the two strings. The original alignment algorithm has been modified to search for the maximum in the last column and row of the matrix. We used the affine map approach as explained in [223] because it has the best biological meaning compared with other methods. In the case of Illumina, we do not allow gaps. MuffinEC does that by selecting the maximum value as described previously, but instead of following a path pre-computed by the default SW algorithm, our software goes straight on a diagonal starting on the maximum value's position. We introduce the concept of consensus *Cons*, which is a matrix for storing a group of reads in a distribution form as shown in table 4.1. A *Cons* counts at each position j how many bases from B are found in all reads overlapping at j. The first read in N is used to create the first *Cons*. While there are reads left unprocessed in N, MuffinEC takes each $r \in N$ and tries to fit it in an existing *Cons*. MuffinEC uses SW to determine the number of differences between a read and *Cons*. Using a user adjustable threshold for the number of differences, the

algorithm decides if r fits in an existing $Cons_i$ or a new consensus for r must be created. In the latter case, the read becomes the seed for a new *Cons*. At the end of this step, MuffinEC ends up with a set containing one or more *Cons*. This part of the software runs in parallel on multiple threads (each thread with its own group), on groups sequentially generated at the step presented in section 4.1.3.

This step is crucial and it has been added not only to tackle the limitations of the fast greedy step, but also for the repetitive and very similar regions. Normally, all the reads that have certain parts in common end up in a big group. As a result there is no problem with isolating some reads that were supposed to be grouped together, they are always grouped together. The problem appears when the reads have common parts but they are actually from different loci of the genome. In that case, the previously presented algorithm searches for the exact MSA of the reads. When there are too many differences (similar regions) or distinct prefixes/suffixes (entry in/exit from repetitive regions), MuffinEC creates subgroups including only those reads which are very similar. The user is free to adjust the quality of the detection by setting different parameters like the maximum number of errors accepted in the MSA and SW penalties.

Error Correction

After the group refining, on the same thread, for a N, MuffinEC corrects the reads within each consensus $Cons_i$. The process iterates through the columns of $Cons_i$ and at each step checks the distribution of bases. Let us consider $\widetilde{b}_j$ to be the correct base at position $j \in Cons_{\mathrm{i}}$ and $Cons_{\mathrm{i,j}}$ all reads overlapping at position $j \in Cons_i$. Furthermore, the program accepts a user defined threshold $p_{\widetilde{b}}$ as the percentage of reads overlapping in j having in the overlapped position base $\widetilde{b}_j$. One can also set the minimum number of

reads overlapping in a position (t_{rmin}, default value **3**). We define $\dot{b}_j$ as the existing base found in the analyzed reads in $Cons_i$ on the j^{th} column. A base $\dot{b}_j$ is considered to be correct if the following holds true:

$$count(\dot{b}_j) := max(count(b_j)), \text{ where } b_j \in B$$
$$\dot{b}_j := \widetilde{b}_j \text{ iff } count(\dot{b}_j) \geq (p_{\widetilde{b}} \text{ x } count(\widetilde{b}_j) \wedge count(Cons_{i,j}) \geq t_{rmin}$$

First, we iterate over the consensus (just once) and correct the bases on the fly. Next, we reconstruct the reads using the (corrected if appropriate) information from the consensus. When a correction occurs, we set the value of the quality (if applicable) for all bases in all reads on the corrected column using:

$$q_{\text{corrected, j}} := \frac{\sum_{h=1}^{count(Cons_{i,j})} q_h[b_{real,h} = \widetilde{b}_{\text{j}}]}{count(Cons_{i,j})[b_{real,h} = \widetilde{b}_{\text{j}}]}$$

Before writing the corrected reads, MuffinEC checks what reads were reverse complemented before correction and, if needed, it converts them to their initial direction. This way, we do not influence the next steps where the direction of a read may be meaningful. Algorithm 2 and 1 summarize the main idea of the group refining and correction algorithms.

The error correction process works best when there is enough coverage on the columns, otherwise the consensus may not be corrected due to lack of information. The insufficient information case is prevalent at the ends of the consensus where MuffinEC may not be able to correct the prefixes/suffixes of some reads.

Input: *N*, *OutFile*
Result: Corrected reads
$ConsArr \leftarrow []$;
$Cons \leftarrow createCons(N[0])$;
$addCons(ConsArr, Cons)$;
foreach $R \in N$ **do**
    foreach $Cons \in ConsArr$ **do**
        if *SW(r, cons)* **then**
            $addRead(Cons, R)$;
        end
        $Cons \leftarrow createCons(R)$;
        $addCons(ConsArr, Cons)$;
    end
end
foreach $Cons \in ConsArr$ **do**
    if $size(Cons) \geq ThreshMinReads$ **then**
        $correct(Cons)$;
    end
    $Reads \leftarrow consToReads(Cons)$;
    $writeR(OutFile, Reads)$;
end

Algorithm 1: The group refining and correction algorithm

Input: *Cons*
Result: Corrected consensus
foreach $Column \in Cons$ **do**
  if *existsMajority(Column, MajBase)* **then**
    foreach $Base \in Column$ **do**
      $Base \leftarrow MajBase$;
    end
  end
  else
    Skip Column
  end
end

Algorithm 2: The error correction algorithm

Calculations

The program follows the C++11 standard and it can be compiled using gcc version 4.7.2 or ulterior. We built and tested it on Linux, but it should work on any other UNIX flavor and even Windows. MuffinEC does not have external third party requirements, except the TCLAP library [224] to process the input parameters, which we ship with our source code. To achieve the best performance we recommend a 64-bit computer with a 64-bit OS. The compiler must support OpenMP 3.0 or greater, because of the use of the OpenMP task directive.

Implementation

MuffinEC uses an unordered map to determine the k-mer spectrum. This data structure has the advantage of fast insertion, deletion and search with an amortized cost of O(1). The error correction step is executed in parallel using a master-slave model,

as depicted in the Supplementary Material. The master thread generates the groups using the greedy approach, while the workers do the costly computation, the alignment and correction. The OpenMP task directive controls the flow of the execution. The error correction step for a group is almost entirely parallel. MuffinEC uses locks to protect the output of the corrected reads in a file.

The memory consumption of the program can be adjusted using C++ templates. We encoded everything that can have an impact on memory consumption as a template. For instance, the k-mer type can be adjusted to use a 32-bit unsigned integer when the k-mer size is less than or equal to 16 or a 64 bit integer otherwise. This has an impact on the k-mer spectrum because it can grow upto tens or hundreds of millions of keys.

Parameters

We make a number of parameters available for our program to keep the execution as flexible as possible. MuffinEC has a technology selection flag that sets most parameters to values suitable for different sequencing technologies. The user can change any parameter, overriding the default values set by any of the technology selection flags. MuffinEC currently supports four technologies and each one can be configured separately. We also provide a generic option, advisable when the sequencing platform is unknown or the software runs on unsupported technologies. This parameter is compulsory, thus the user must set it to one of the five available values, namely *illumina, 454, ion, pacbio, generic*. The profiles for different technologies are shortcuts that set internally the rest of the parameters to values determined to be the best for a certain technology.

One important setting is the maximum number of errors accepted by the SW regrouping. It can filter out those reads that are not related to the others in a group obtained by the step from section 4.1.3. An inadequate value for this option translates into

the admittance of a read that can destroy a subgroup's consensus, rendering the whole correction process useless. This parameter sets a threshold for the number of differences between a read and a consensus. If the number of differences exceeds it when SW executes the backtracking part, the algorithm does not accept the read in the current subgroup.

A third parameter with a strong influence on the results is the number of bases having the same nucleotide when executing a correction of a column in a consensus. If the value is too low, MuffinEC will change a potentially valid base into an erroneous one. This case appears when the coverage on a column is low. In sub section 4.3.1.4, we present the results of our corrector with various values for this parameter. The default value is 51%.

Finally, the default value of the k-mer length (user adjustable) is 21 (the same default value as in Coral and HECTOR). This value is the most appropriate in the case of bacterial genomes, which are typical cases where error correction is applied. The user is free to adjust this value as per his/her requirements.

MuffinEC accepts the following parameters listed by running *./MuffinEC –help*:

```
      # ./MuffinEC -a <string>|-q <string> --454 | --illumina
| --pacbio | --ion | --generic [--allowIndels] [--gencon]
[--trimQualThresh <positive number>] [--mincomkperc <decimal
percentage>] [--errorPercOverlap <decimal percentage>] [-k
<natural number>] [--maxsn <positive number>] [--minsn <positive
number>] [--errsmaxperc <decimal percentage>] [--algapext <real
value>] [--algap <real value>] [--almat <real value>] [--almis
<real value>] [-m <decimal percentage>] [-p <positive number>]
-o <string> [--] [--version] [-h]
Where:
-a <string>, --fasta <string>
(OR required) The full path of the FASTA file containing the
reads to be corrected
```

```
-- OR --
-q <string>, --fastq <string>
(OR required) The full path of the FASTQ file containing the
reads to be corrected


--454
(OR required) The input data is from 454
-- OR --
--illumina
(OR required) The input data is from Illumina technologies
-- OR --
--pacbio
(OR required) The input data is from Pacific Biosiences
-- OR --
--ion
(OR required) The input data is from Ion Torrent
-- OR --
--generic
(OR required) use this option if you don't want to set/know the
source technology


--allowIndels
Add this flag to allow indels when creating the consensus of a
group of related reads

--gencon
Add this flag to generate the contigs for the consensus

--trimQualThresh <positive number>
Value marking the minimum accepted quality score for a base at
the ends of the reads such that the position won't get axed
```

```
--mincomkperc <decimal percentage>
Percentage of overlapping for two reads, calculated from the
smaller between the two

--errorPercOverlap <decimal percentage>
The percentage of errors found in the overlap region of two
reads compared by the fast gapped kmer algorithm

-k <natural number>, --kmerlen <natural number>
the length of the kmer used to do the coverage and to create the
neighborhood

--maxsn <positive number>
Maximum size of a neighbourhood, created by the grouping method
and which will be further processed by the correction mechanism

--minsn <positive number>
Minimum size of a neighbourhood, created by the grouping method
and which will be further processed by the correction mechanism

--errsmaxperc <decimal percentage>
The max number of errors accepted by the Smith Waterman
algorithm when comparing the distance between a read and
a subgroup; if the total number of differences between the
read and the subgroup are greater than the limit set by this
parameter the read won't be added to the subgroup

--algapext <real value>
Aligner gap extending score

--algap <real value>
Aligner gap opening score
```

```
--almat <real value>
Aligner match score

--almis <real value>
Aligner mismatch score

-m <decimal percentage>, --percentMajorityDist <decimal
percentage>
The percent of reads which must have the same base on a column
of the consensus to apply a correction

-p <positive number>, --threads <positive number>
number of threads used by the program

-o <string>, --output <string>
(required) the name of the output file containing the corrected
reads

--, --ignore_rest
Ignores the rest of the labelled arguments following this flag.

--version
Displays version information and exits.

-h, --help
Displays usage information and exits.


MuffinEC Multi-technology Indels-Aware NGS Error Correction
Software
```

The next section discusses the robustness of the chosen settings and/or methods used to calculate the default values. The rest of

Table 4.2: Experimental Datasets; the last two columns show number of reads in the original dataset (penultimate column) and filtered dataset respectively (last column)

Dataset	Genome	Accession #	NCBI Ref.	Platform	Genome Size	Read Length	Coverage	# Reads Orig.	# Reads Filt.
D1	*E. coli*	SRR022918_1	NC_000913	Illumina	4 639 675	47	72	7 204 315	4 134 786
D2	*E. coli*	SRR000868	NC_000913	454	4 639 675	37 – 385	12	230 517	223 055
D3	*E. coli*	ERR039477	NC_000913	Ion Torrent	4 639 675	16 – 107	7	390 975	366 969
D4	*E. coli*	SRR620425	NC_000913	Ion Torrent	4 639 675	6 – 395	155	4 237 734	3 838 047
D5	*E. coli*	SRR1284073	NC_000913	Pacific Bioscience	4 639 675	22 – 29453	139	163 464	29 426
D6	*S. aureus*	SRR022866_1	NC_003923	Illumina	2 860 755	76	339	12 775 858	7 475 923
D7	*D. melanogaster*	SRR035606	RELEASE_5_48	454	119 029 979	35 – 609	<1	239 192	134 312
D8	*E. coli*	SRR1640170	NC_000913	Illumina	4 639 675	100	523	24 298 709	23 334 662
D9	*A. muciniphila*	SRR073384	NC_010655.1	454	2 664 102	53 – 225	15	353 852	325 853
D10	*S. enterica*	SRR1183991	NZ_CP007422.1	454	4 734 890	57 – 1601	110	429 755	176 935

the parameters are described in the documentation file accompanying the software.

Results and Discussion

We tested MuffinEC against Coral, HSHREC, Fiona and HECTOR (please refer to chapter 2 for a presentation of each corrector). All of these tools support indels and were downloaded from their websites as source code or compiled if possible. We selected various datasets from the literature for all the supported technologies. The chosen datasets (Table 4.2) try to cover most of the current technologies and they pose complementary challenges (length, coverage, error rates, error types etc.).

We chose *E. coli* as our model organism because there are many well studied datasets for it. *E. coli* appears in almost all the literature about error correction. Datasets D1, D2 and D3 were chosen because they appear in [15, 1]. D1 can also be found in other papers that address Illumina data [54]. We selected the second Ion Torrent dataset, D4, because of its size. To analyse the performance of the correctors on PacBio data, we selected an *E. coli* dataset (D5). D6 appears in Coral's article, therefore we included it in ours for the sake of comparison. D7 is an example of a dataset

with indels and from a larger organism. D8 is the largest Illumina dataset, being an excellent benchmark for testing the behaviour of correctors in the case very high coverage datasets. D9 and D10 are two examples of 454 data (cases in which we can test all the correctors in our benchmark, including HECTOR) from another two organisms.

The selected datasets also vary a lot in coverage. This variation should test the corrector's behaviour in many situations, especially when the overall coverage is very low e.g. D2. D7 is a special example because it is the result of transcriptome sequencing. As a result, the information in reads only targets those genes being expressed at the sequencing time. This is an additional test where the reads are isolated and the corrector cannot expect a contiguous alignment of sequences. Even though MuffinEC has some limitations when it comes to very low coverage as described in section 4.1, it still manages to win against the other algorithms in the tests. Coral, which is supposed to include a read in more than one group, is quite slow and its correction performance not on par with MuffinEC as we shall see later in this section.

We performed the tests on a 64bit Ubuntu 14.04 machine with an Intel Core i7-3930K CPU @ 3.20GHz and 64 GB RAM.

Testing Methodology

Testing the accuracy of correction tools in real datasets proves to be a challenge because the positions of errors in sequencing data are unknown. There are numerous methods in the literature to evaluate correction performance, for example [9, 1, 107]. We chose the method used in [1] because it is widely employed in the literature. The first step of this method is filtering via a mapping algorithm (bowtie2 [129] in our case) to determine and discard reads not mapping the reference genome or mapping to it multiple times. This approach may eliminate many tricky cases by filtering

the input dataset (as stated in [107]). For the PacBio datasets, we used blasr [225] (specifically designed for PacBio) because bowtie2 was unable to map enough reads to generate a meaningful filtered dataset for correction. A script filtered out the mapping reads from those not mapping and generated a new dataset with only the uniquely mapping reads. Eliminating these cases minimizes the risk of misclassification [15]. The parameters used in case of bowtie2 and blasr are the default ones, predefined by the aligner. The alignment algorithms are not perfect and may misalign some of the reads. To mitigate this, we align the original reads once and use the found locus on the genome to generate the alignment for both the original and corrected (as we shall see next). If case of misalignment, the results for that particular read are erroneous. In this case (since the data is real and no one can possibly know for sure the location for each read) the only way to overcome this is to use other metrics to validate the results. We understood this risk, but from our check on samples of data the aligners are pretty good at not making many errors. Nonetheless, we include in our set of metrics the percentage of reads which align after correction.

The next step was to feed the filtered dataset to the programs from our test. To assess the capability of each candidate, we defined the following metric (gain) [1, 2]:

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

Similarly, two more metrics used in [98] are Sensitivity

$$SE = \frac{TP}{TP + FN}.$$

and Specificity

$$SP = \frac{TN}{TN + FP}.$$

In the previous formulae, the variables represent:

- TP (true positives) existing errors that are corrected.
- TN (true negatives) correct bases left unmodified.
- FP (false positives) correct bases that are wrongly considered being faulty.
- FN (false negatives) erroneous bases left unmodified.

We consider r_o as the original, uncorrected read and r_c the same read obtained after correction. To calculate TP/TN/FP/FN, the evaluation procedure uses the location in the reference genome from the alignment of the original read. We used a version of the NW algorithm from the NCBI C++ Toolkit [226], allowing free gaps at the beginning and the end of the alignment. The scoring method mapped r_o and r_c, on the region previously determined by bowtie2 or blasr, respectively. We avoided using the distance calculated by the aligner to be sure that we used the same scoring scheme for both the original and corrected datasets. Using the **CIGAR** string (Matches/Mismatches marked separately) produced by NW, the evaluation procedure compared the number of differences between the two reads and incremented $TP/TN/FP/FN$ accordingly.

Table 4.3 presents the results of the evaluation. All programs run with the default parameters, unless stated otherwise. The greedy approach is more memory efficient and faster compared with most cases. HECTOR performed better from a resource consumption standpoint in all 454 tests, but it did not correct as well as the others. From the perspective of gain and sensitivity, MuffinEC managed to obtain a better score in all cases. On D5, MuffinEC was slower than HSHREC but showed much better gain and sensitivity, and its memory consumption was lower.

In contrast to HSHREC, MuffinEC remains constant in correcting the datasets without any negative spikes (HSHREC obtains

negative scores on D5 and D6). Coral failed with a memory access error for the PacBio dataset (D5). This is understandable considering that PacBio was not on the market as a viable solution when Coral was released.

HECTOR, failed to run with the defaults arguments, throwing the following error: "void ParaKmerEC(ProgramOptions&): Assertion 'false' failed". We informed the authors about the problem and, following their advice, we tried the corrector only with FASTA files, removing the quality scores from the FASTQ files.

HSHREC failed to run on D7. It constantly asked for a smaller value for the strictness parameter. We tried (without success) all natural values from 7 down to 1. For this dataset that posed problems for HSHREC, we ran all the other algorithms with a value of 25 for the k-mer length (the value obtained by using the formula in Subsection 4.2.2). We chose this value because of the larger size of the genome of the organism being analyzed. The number of threads for each program was set according to the number of CPU cores available.

We measured the value of the Resident Set Size (RSS) memory which is a closer estimate of the real memory consumption. The difference in memory consumption measurements in previous papers is because Coral seems to allocate a great chunk of Virtual Memory (VM), but the real memory used by the program varies with the total RAM available in a system. Therefore, VM is stable across different setups, but RSS varies greatly. For datasets D1, D2 and D6 the memory consumption also appears in [15]. For datasets D2 and D3 more details about memory consumption on older Linux systems appear in [1]. We let Coral use as much memory as it wanted to obtain the maximum performance.

To help the correctors and improve the correction, we set the appropriate technology flags (if available) for each dataset. We ran Coral with the flags corresponding to both 454 and Illumina. For the datasets not officially supported, Coral used default values,

without any technology flag. For HSHREC, we had to write a testing loop to determine the maximum working value of the strictness parameter. Without this, HSHREC would not successfully execute on some datasets, asking for a lower value of the aforementioned parameter (default 7). Fiona set the "–sequencing-technology" flag to the corresponding technology for each dataset being corrected. For the PacBio dataset, Fiona did not use any technology flag at all. Our program also utilized a technology flag, each time being set to the suitable value for the analyzed dataset, namely one of the values presented in section 4.2.2.

Another aspect is the input/output type we used for the three programs. HSHREC and HECTOR do not work with FASTQ files, thus we had to remove the quality scores for the inputs. Because Coral, Fiona and MuffinEC do not have this limitation, we decided to test them using FASTQ files.

Resource Consumption Testing

We used GNU time command instead of the builtin time command because the former offers more information about the running process. From the output generated by this command, we also extracted the memory consumption given under the name of "Maximum resident set size". This measurement informs us over the maximum memory used by a process during its execution. This is very important in contrast with the memory occupied by the process at the end of its execution because it gives an accurate view over the real need of RAM. For instance our program uses C++ STL unordered maps. They grow as you insert elements and shrink when you delete elements. There is a feature implemented called rehash which when executed will re-arrange the elements inside the map. This will usually result in a decrease in memory usage. Since we delete elements during the k-mers removal stage, we call this method at the end of the cleaning process to shrink the size. Now

Table 4.3: Error Correction Testing Results

Dataset	Algorithm	Sensitivity	Specificity	Gain	Memory (GB)*	Time (mins)
D1	Coral	0.74	0.99	0.74	11.2	3.28
	HSHREC	0.68	0.99	0.68	8.3	19.48
	Fiona	0.54	0.99	0.54	2.4	8.33
	MuffinEC	**0.82**	0.99	**0.82**	**1.7**	**1.83**
D2	Coral	0.78	0.99	0.75	8.6	3.10
	HSHREC	0.81	0.99	0.80	10.8	5.88
	HECTOR	0.53	0.99	0.53	**0.2**	**0.21**
	Fiona	0.83	0.99	0.83	1.0	2.28
	MuffinEC	**0.86**	0.99	**0.84**	0.8	0.96
D3	Coral	0.78	0.99	0.77	9.3	0.51
	HSHREC	0.78	0.99	0.78	7.1	3.93
	Fiona	0.75	0.99	0.74	1.0	2.76
	MuffinEC	**0.94**	0.99	**0.94**	**0.5**	**0.25**
D4	Coral	0.74	0.99	0.67	20.5	157.41
	HSHREC	0.28	0.99	0.03	25.5	90.38
	Fiona	0.76	0.99	0.77	5.3	40.28
	MuffinEC	**0.86**	0.99	**0.84**	**4.2**	**14.01**
D5	Coral	-	-	-	-	-
	HSHREC	0.0004	0.99	-0.003	25.5	**40.66**
	Fiona	0.001	0.99	0.0006	39.2	244.43
	MuffinEC	**0.08**	0.99	**0.08**	**20.2**	83.58
D6	Coral	0.63	0.99	0.63	21.2	53.38
	HSHREC	0.30	0.97	-0.31	26.4	70.20
	Fiona	0.69	0.99	0.69	6.0	19.15
	MuffinEC	**0.71**	0.99	**0.71**	**3.7**	**5.33**
D7	Coral	0.25	0.99	0.24	13	2.83
	HSHREC	-	-	-	-	-
	Fiona	0.12	0.99	0.11	0.9	2.33
	HECTOR	0.11	0.99	0.11	**0.1**	**0.10**
	MuffinEC	**0.40**	0.99	**0.38**	0.8	1.01
D8	Coral	0.94	0.99	0.94	26.1	319.4
	HSHREC	0.61	0.99	0.48	34.3	324.68
	Fiona	0.95	0.99	0.95	22.5	52.48
	MuffinEC	**0.98**	0.99	**0.97**	**8.4**	**29.91**
D9	Coral	0.78	0.99	0.77	12.8	1.05
	HSHREC	0.38	0.99	0.37	6.9	5.31
	Fiona	0.57	0.99	0.57	1.0	2.23
	HECTOR	0.23	0.99	0.23	**0.1**	**0.10**
	MuffinEC	**0.93**	0.99	**0.93**	0.5	0.41
D10	Coral	0.01	0.99	0.01	17.7	20.86
	HSHREC	0.09	0.99	0.06	23.3	28.68
	Fiona	0.09	0.99	0.09	1.6	14.96
	HECTOR	0.01	0.99	0.009	**0.3**	**0.73**
	MuffinEC	**0.95**	0.99	**0.95**	1.2	2.86

*RSS memory value measured by GNU Time

let us consider an example in which we keep adding elements till our map reaches 20 GB. We do the cleaning and rehash and we end up with a 16 GB object. It will be totally wrong to say that our program uses just 16 GB. On a machine with just 16 GB of RAM the system will kill the process and the program will fail. The following snippet is an example of the time command output:

The following is an example of testing procedure used for all programs:

```
# Using the following parameters values:
In file:  /home/asalic/test/ecoli/illumina/SRR1640170.fastq
Out file:  /home/asalic/test/ecoli/illumina/SRR1640170_MuffinEC.fastq
Use OMP: yes
Number of threads to be used for error correction:  3
Alignment match:  2
Alignment mismatch:  -1
Alignment gap:  -3
Alignment gap extension:  -1
Max percentage errors accepted in SW: 0.14
kmer Size:  21
MinNum Percentage Common Kmer:  0.5
Percentage Errors in the Gapped Kmer Overlap :  0.02
Trim Bases With a Quality Below :  53
Generate Contigs:  false
Minimum Overlap percentage:  0
Percent Common Bases Column Consensus for Valid Correction:  0.51
Min Size Neighborhood:  3
Max Size Neighborhood:  500
Selected Profile:  Illumina
Allow indels during consensus generation:  false
It's time to correct the errors
Count kmers
35350474 different kmers found
Qualities between 35(#) and 74(J)
Create kmers histogram
```

```
Filter kmers
Remove kmers with low level of expression
Cut histo at 3
Alloc space for IDs of reads containing left kmers
Group reads
Unique kmers left after removal stage:  9613863
Start error correction
Reads processed/total:  2429342/24292363
Reads processed/total:  4858554/24292363
Reads processed/total:  7287865/24292363
Reads processed/total:  9717270/24292363
Reads processed/total:  12146603/24292363
Reads processed/total:  14575496/24292363
Reads processed/total:  17004909/24292363
Reads processed/total:  19433932/24292363
Reads processed/total:  21863163/24292363
Reads processed/total:  24292361/24292363
Number of failed to correct reads:  3
Correction process has ended!
Number of neighborhoods:  260392
Largest neighborhood:  501
Number of reads associated with groups:  24298709
Total number of reads in the file:  24298709
Number of skipped short reads:  0
Number of skipped low qual reads:  6346
Number of oversized neighborhoods:  1754
Number of greedy falbacks/total number of greedy comparisons:  38447/312281
happy ending after:  2761 secs
total size of corected contigs 21151025
Groups with 1 elem:  142690
Groups with 2 elems:  0
Num reads in groups smaller than 3:  0
Command being timed:  "/home/asalic/tmp/MuffinEC-1.1/build/MuffinEC
```

```
--illumina -p 3 -q /home/asalic/test/ecoli/illumina/SRR1640170.fastq
-o /home/asalic/test/ecoli/illumina/SRR1640170_MuffinEC.fastq"
User time (seconds):  6129.36
System time (seconds):  95.91
Percent of CPU this job got:  225%
Elapsed (wall clock) time (h:mm:ss or m:ss):  46:01.35
Average shared text size (kbytes):  0
Average unshared data size (kbytes):  0
Average stack size (kbytes):  0
Average total size (kbytes):  0
Maximum resident set size (kbytes):  8761176
Average resident set size (kbytes):  0
Major (requiring I/O) page faults:  0
Minor (reclaiming a frame) page faults:  7370293
Voluntary context switches:  2327
Involuntary context switches:  550818
Swaps:  0
File system inputs:  0
File system outputs:  17042768
Socket messages sent:  0
Socket messages received:  0
Signals delivered:  0
Page size (bytes):  4096
Exit status:  0
```

We had to patch the GNU time command because as a result of a bug it incorrectly reports the *Maximum resident set size (kbytes)*. The line 395 from *time.c* must be changed from:

fprintf (fp, "%lu", ptok ((UL) resp->ru.ru_maxrss));

into

fprintf (fp, "%lu", (UL) resp->ru.ru_maxrss);

In the case of the Java program, we decided to use the same GNU time command to measure the memory consumption. The main reason we chose this path over specialized methods like YourKit

Java Profiler [227] or EJ-Technologies JProfiler [228] is that we need an overview over the full memory requirement of the program. We could measure the memory consumption for only the Java program and disregard the overhead of the Jvm but it wouldn't have offered a realistic view. The whole purpose of the memory section is to give an accurate as possible measurement of the RAM needs so the whole program works. Again, if Hshrec is using 16 GB as a inside process in the JVM but the overall JVM requires 18 GB, a machine with only 16 GB won't be able to execute the program and probably will have to kill the process.

To test Hshrec we used Oracle Java 1.7.0_55. We set the maximum memory of the JVM to be the whole quantity of RAM available on the testing machine. We left the minimum memory with its default value.

Scalability

Nowadays, the CPU is limited in performance as a single unit. There is no method to just increase the frequency and the execution speed overall. As a result, the only method to dramatically improve speed is to use more than one execution core, to split the problem in tasks that can be executed at the same time. Therefore, the concept of parallelism has become ubiquitous in Computer Science and today even our mobile phones have multi-core CPUs. We built MuffinEC to take advantage of the local parallelism by the means of OpenMP. Figure 4.3 depicts the dramatic reduction of the total running time of our application when using between one and six core. the previous subsection contains a trial run for three cores where the output of MuffinEC and of the time command are listed. The same figure also contains the memory consumption with different number of cores. We used KB together with a very narrow interval (from 8,760,000 to 8,770,000) to mark the potential differences as well as possible. Please bear in mind that the results

in GB from table 4.3 are obtained by dividing by 1024, not 1000. The analysis is performed on D8 (without filtering), the largest dataset from our benchmark. The selection of D8 was based on the assumption that using the highest quantity of data we are able to counteract the effects of external factors like the OS executing a short lived job besides our program. Furthermore, using the largest dataset, we are able to see if the algorithm is susceptible to dramatic increases in memory consumption when more workers are added. Figure 4.3 shows a insignificant variation (maximum of approx 2.3MB from 8.4GB when using six cores instead of one, an increase of 0.02%) in the memory consumption, demonstrating the stability of scalability.

Profiling

In this section we continue our exploration of parallelism. We concentrate on the time spent in different portions of the application. For the test, MuffinEC ran with 6 threads under the supervision of Intel VTUNE 2016 on Ubuntu 14.04. Intel's profiling tool generated a summary with all the hotspots of the application. We detected three methods corresponding to three distinct steps from figure 4.1, namely the k-mer counting and histogram, the initial read grouping, and the group refining and error correction. Figure 4.4 shows that the weightiest part is the actual group refining and error correction. This is the main reason for us to choose this part as the most important to execute in parallel. Once multiple cores are allowed, the time spent in the aforementioned time consuming parts drops significantly as depicted in figure 4.5.

Parameter Robustness

We followed multiple approaches to determine the optimal values of MuffinEC's parameters. First, we searched through the literature

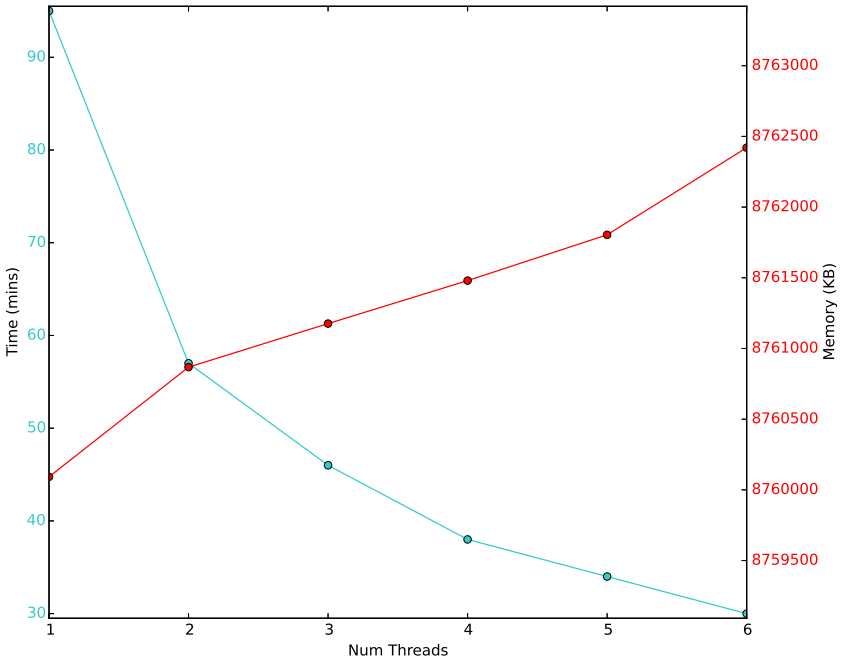


Figure 4.3: Execution time and memory consumption for multi-core execution.

for similar parameters. For instance, the optimum k-mer size has been debated in many published articles. The next list summarizes the formulas for determining the k-mer size from different sources:

- $k = \log_4 200N$ [9, 12]
- $k \geq \lceil \log_4 N \rceil$ [15]
- $4^k > N$ [98]
- $k = \lfloor l/6 \rfloor$ [11]

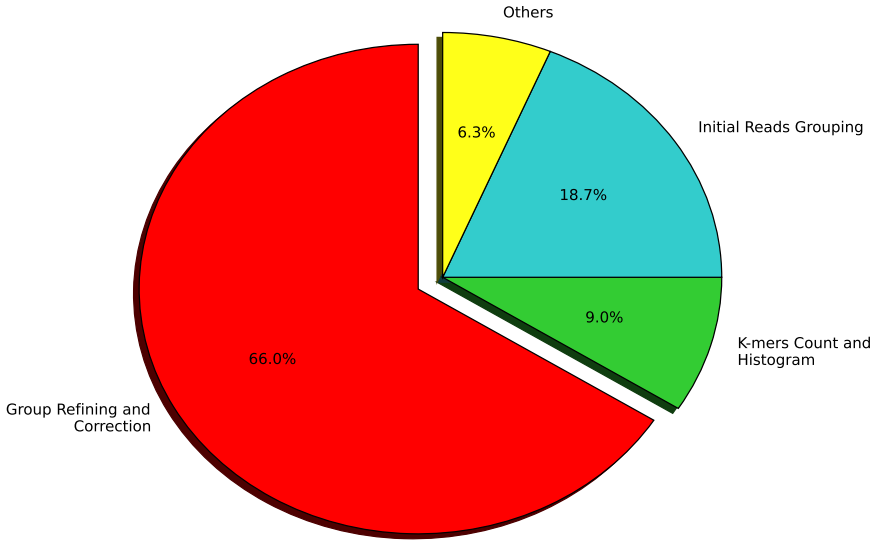


Figure 4.4: Profiling using one core

- $k = \log_4 2Np^{-1}$; $p = 10^{-4}$ [110]
- $N_s/4^k \leq 0.0001$ [109]

where N = genome length, l = read length; p = probability that a random k-mer appears in a random string of length N, using the alphabet {A, C, G, T}; N_s = number of unique solid k-mers

Five out of six papers chose a variation of the same general formula, hence we selected $k = \log_4 2Np^{-1}$ with $p = 10^{-6}$.

Next, we used empirically derived values from the literature. This is the case of error correction overlap error rate In case of Il-

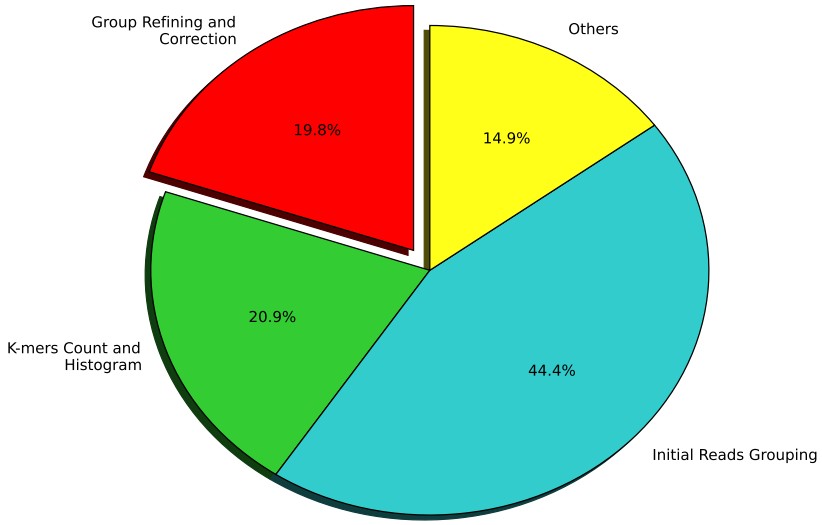


Figure 4.5: Profiling using six cores

lumina, [61] describe the substitution errors for various sequencers. For instance, the Genome Analyzer® produces many G → T and C → A substitutions [9, 63]. Normally, the percentage of errors increases towards the end of a read (the 3' end) [62, 9]. The estimated error rate for Illumina is between 1 and 2.5% [64, 65, 9], hence we selected 2% as the default for this technology.

Finally, for the parameters that are confined to our own approach, we performed an experimental analysis of their behaviour and robustness. We present herein a study of the interdependence between two very important settings for our program. Due to the

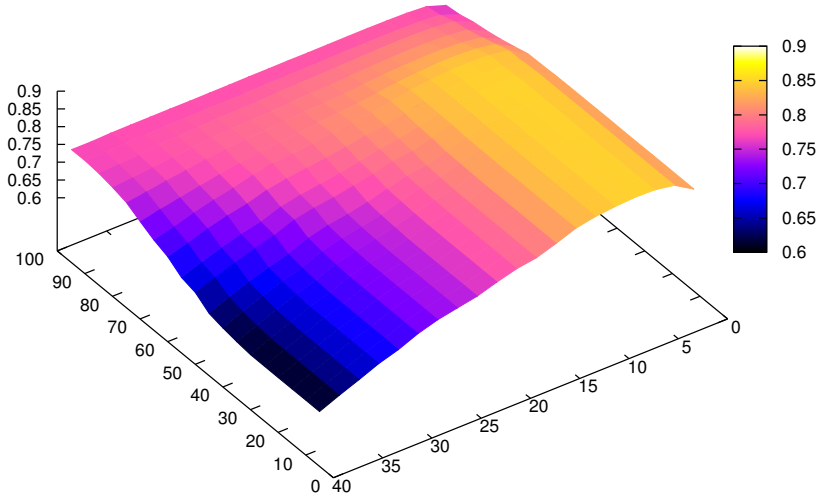


Figure 4.6: Robustness analysis for D2, varying two parameters.

high number of parameters accepted by our system, we focus only on the extensive testing of two key parameters with a very high impact on the final result. One of the parameters is the SW error rate and the other is the percentage of reads having the same nucleotide in a consensus position to be considered correct. Fig 4.6 presents the results of multiple executions for the gain metric with dataset D2. We selected for our empirical evaluation the percentage of bases that must agree in a column to perform correction and the maximum percentage of differences allowed when comparing a read against a subgroup's consensus to accept that read in the subgroup. To execute the tests, we varied each parameter between the limits of a meaningful interval, i.e. for the former from 5% to 95% and for the latter from 2% to 40%. Using this type of approach,

we determined the optimum values for our parameters by testing them on a representative set of data. As a result, it is clear that MuffinEC is quite stable with respect to the parameters and the high values for the gain are situated somewhere near the center of the 3D shape. We went for extreme values for the percent of the majority to show that although this parameter has a strong impact on the outcome, our program remains quite stable and still produces meaningful results (as long as the gain is not negative, some correction is still being made).

Short Aligning Results

Finally, to complete our analysis of correction accuracy, we computed the reduction of non-mapping using bowtie2 (the same aligner as in the Gain/Specificity/Sensitivity filtering stage). To make the test as even as possible for all programs, we decided to use FASTA for all correctors as it is supported in all cases. For this test, the correctors ran on the full dataset rather than the filtered one. We compared the alignment results for the corrected data against the results for the original data using bowtie2. MuffinEC managed to reduce the number of non-mapping reads by 15% (Fig 4.7), the highest percentage among all the tested programs.

Assembly Results

To complement the previous widely used metrics, we decided to add results and discussion about the influence of error correction on assembly. We present these detailed results obtained from the full dataset D9. We selected this dataset because it is the best example where we can demonstrate that the gain/sensitivity/specificity metric is sufficiently accurate to compare the correctors. Furthermore, D9 is from a 454 sequencer, supported by all correctors (including HECTOR) and a good example of a set having all types

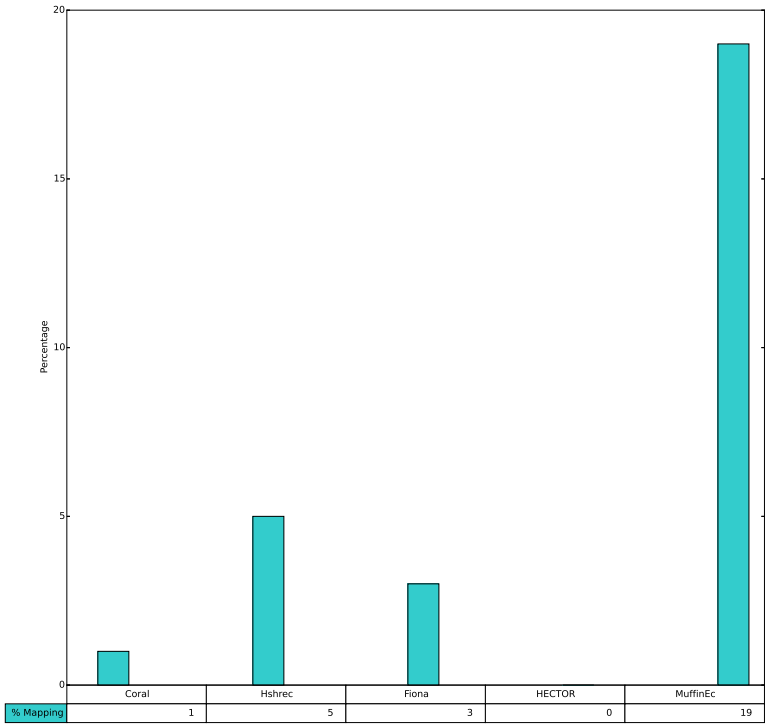


Figure 4.7: Percentage of mapping reads from the NON-mapping ones after correction as reported by bowtie2; results with the FASTA output; higher is better.

of errors. We chose Newbler [229], Roche's own assembler, as it should produce the best results with 454 data (its author is the company that manufactures the 454 sequencer-family) [230]. The assembler test supports and reinforces the usability of the error correction for a real use case. To make this test as close to reality as possible, we decided to skip the preprocessing step used in

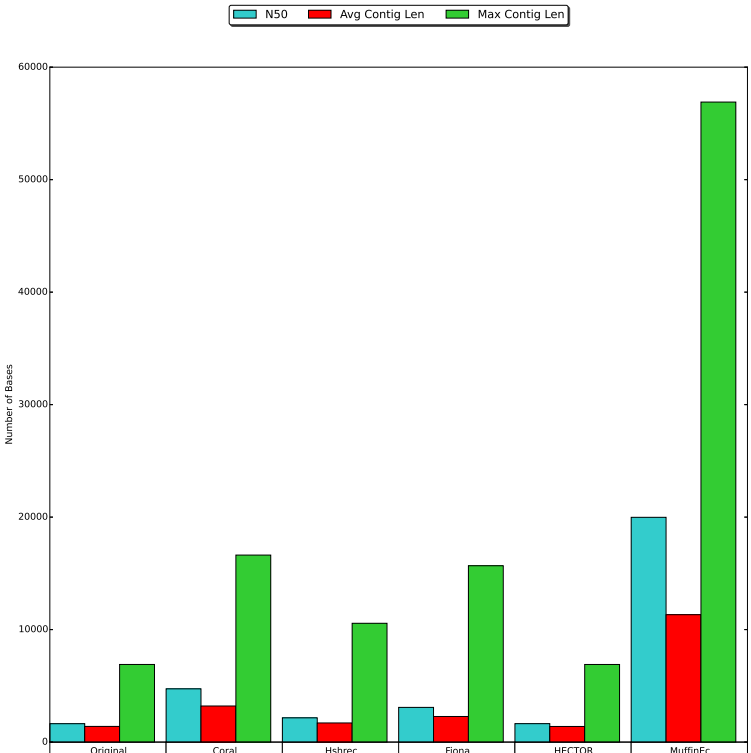
the case of the gain/sensitivity/specificity and feed the full original dataset (as extracted from the SRA) to each corrector in our test. We depict in Fig 4.8 the average and maximum contig size along with N50 [231]. Additionally, we present the count of large contigs (default 500bp as considered by Newbler) and all contigs. This metric is very useful for observing the fragmentation of the final result.

As seen in Fig 4.8 and Fig 4.9, MuffinEC manages to improve the assembly results both in terms of contig size and total contig number. HSHREC's output does not respect the FASTA standard and inserts a space between the read id's reserved first character, '>', and the description string that follows before the new line. As a result, we had to modify the output to successfully execute Newbler with HSHREC's output.

For the reader's convenience, we also included the results obtained from the original uncorrected dataset. By comparing the behavior of an assembler with raw versus corrected input, one can assess the assembly software's degree of error tolerance. Regarding D9, it is clear that assembly by Newbler improves when it is fed corrected data regardless of the corrector. However, MuffinEC boosts the assembly performance by a large margin compared to the results obtained using the alternatives in our test.

Unknown Bases

Table 4.4 contains statistics regarding unknown bases for the original, filtered and corrected datasets. We chose one Illumina dataset and one 454 dataset from two different organisms. Overall Coral was better than both HSHREC and Fiona, while MuffinEC was superior in all cases. All the software in our test managed to remove a large percentage of the unknowns.



	Original	Coral	Hshrec	Fiona	HECTOR	MuffinEc
N50	1635	4734	2156	3085	1638	19982
Avg	1394	3209	1702	2276	1389	11325
Max	6901	16626	10562	15677	6900	56898

Figure 4.8: Newbler's metrics; larger is better (longer contigs means that the assembler was able to build longer stretches from the original genome)

Resource Demands

A top priority for MuffinEC is to keep both the computational cost and memory consumption low as the amount of data increases. The main challenge in memory consumption is the storing of the k-mers count and the k-mer - reads map. These two structures need to be

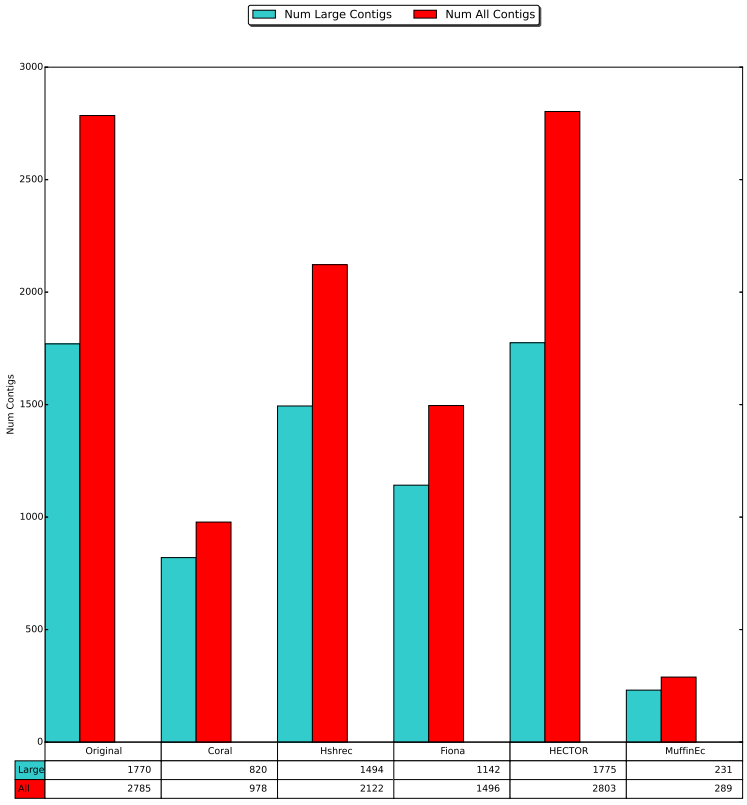


Figure 4.9: Number of large contigs and all contigs as reported by Newbler; Shorter is better (fewer contigs, closer to a final genome)

in RAM to obtain good performance.

Our solution exploit multicore processors very well thanks to a multi-threading implementation. However, the matrix used by the SW algorithm consumes more memory when running in parallel because each thread needs its own matrix instance. Furthermore, to avoid memory fragmentation and many allocation/deletes, the software assigns a static, re-sizable matrix to each thread when the

Table 4.4: Dataset Statistics with Respect to Unknown Bases

Dataset	Algorithm	Reads with One or More N's	Num. N Bases
D1	Original	84 710	2 275 945
	Filtered	3 312	4 527
	Coral	927	1 750
	HSHREC	1215	2 346
	Fiona	1288	2 194
	MuffinEC	**484**	**856**
D7	Original	31 465	100 735
	Filtered	15 238	29 532
	Coral	9 063	20 019
	Fiona	9 415	21 200
	HECTOR	15 238	29 532
	MuffinEC	**7 198**	**15 120**

program starts and does not free the memory held by the matrix until the program ends.

The first two steps described in section 4.1 are sequential. The genuinely parallel part is the correction of the groups, which typically takes a significant amount of the whole running time. This happens when the second step of the algorithm does not group the reads well. Then, the refining part has more comparisons to perform.

We measured the RAM requirement and the total time spent for Coral, Fiona and MuffinEC using the GNU time command. In the case of HSHREC, we decided to apply the same testing methodology as for the others. We measured the memory consumption of the Java process, as explained in more detail in the Supplemental Materials. HSHREC artificially alleviates the increased memory consumption by working only with FASTA files (it does not load

the quality scores) and by not preserving the original tags for each FASTA entry. Table 4.3 contains the resource consumption data. The fastest corrector by far in our benchmark, HECTOR, suffers from a reduced correction rate when compared to the selected competition.

Discussion and Conclusion

MuffinEC is an error correction program for de Novo assembly capable of handling all types of errors in four current sequencing technologies, namely, Illumina, Roche 454, Ion Torrent and PacBio (plus generic support for future ones). It is implemented in C++11 with OpenMP and accepts a wide range of parameters. It tackles the problem in five steps:

- k-mers counting and histogram generation,
- initial read grouping by one common k-mer,
- greedy grouping by number of k-mers,
- refining using the SW algorithm of the greedy groups obtained at the previous step and subgroups generation when needed,
- the actual error correction of the (sub)groups.

This program works on commodity hardware and has no limitations in regard to to variable read lengths or uncalled bases.

In our experiments MuffinEC obtained better error correction results in all cases against selected competition from the literature. We tested it against five correctors using four different correction quality metrics:

- percentage of corrected errors in the form of gain, specificity and sensitivity,
- percentage of short alignment after correction compared with the percentage before,
- assembly metrics,
- unknown bases count before and after correction.

We also verified the robustness of the default values for the parameters accepted by MuffinEC. Because our implementation achieves higher speed and lower resource consumption in most testing cases, it should be suitable for large datasets, more complex genomes and even new technologies. Section 4.3.1 contains a detailed analysis of the resource consumption and the hotspots in the application.

CHAPTER 5

MuffinInfo - NGS Information Extractor

An earlier proof version of the software has been presented as a poster as: **Alic AS.**, Blanquer I. MuffinInfo: HTML5 statistics extraction system from FastQ/Fasta/Sam files. *In:* F1000Research (ISCB/ECCB '15). 2015. DOI: 10.7490/f1000research.1110049.1

Part of this chapter has been published as: **Alic AS.**, Blanquer I. "MuffinInfo: HTML5-based statistics extractor from Next Generation Sequencing data". *In:* Journal of Computational Biology (In Press). 2016

In this chapter we introduce MuffinInfo, a HTML5 information extraction tool from NGS data. One of the first reason for developing this tool is that many NGS analysis applications (like assembly, error correction, variant calling etc.) depend on programs that are not capable to determine the best parameters by themselves. Guidelines may exist to help the user to select a suitable execution configuration for a NGS analysis application, but the deep knowledge of the sequencing data characteristics can help the user tremendously. It is much more efficient to have some insights into the data than to go blindly and try different combinations of parameters for the applications. Information extractors like MuffinInfo and FastQC are designed to aid the user in his/her work with the data. Our work aims at helping the selection process of the proper configuration for NGS analysis applications and can also give clues about the results of different tools. For instance, the k-mer distribution can come in handy for a user utilizing error correctors that are based on k-mers and are unable to automatically select the best k value. The problem is that on the one hand a value too low for a k can result in a very high multiplicity for the resulting k-mers which basically hides the errors. On the other hand, a value too high for k can generate too many unique k-mers, hence the errors cannot be detected and corrected.[9]

We selected HTML5 because it enables the scientist to perform his/her work from whatever device (phone or PC) and whenever in the world (online or offline). For the sake of simplicity, we will refer to HTML5 as a general term which includes any combinations of the underlying technologies: Javascript, CSS and HTML. With so many existing hardware (laptops, phones, tablets; Intel x86 and x64, ARM, SPARC) and software (Windows, Mac, Linux, Android, iOS) options, a modern application should be able to run on as many platforms as possible, regardless of the underlying environment. However, despite the hardware currently being fast enough to handle portable languages like Java and Python for in-

tensive computations, parsers and virtual machines are generally available only for a limited number of platforms. HTML5 offers the possibility to write once and then run on a great number of hardware and software platforms. We also have to consider the increasing penetration of internet (from TVs to smart labs) which is an important catalyst for further development of web technologies. this means that in the foreseeable future HTML5 will continue to grow and be available on newly created devices (as everything nowadays seems to be connected to the internet). Additionally, we needed a software supporting easy addition of new statistics. Owing to its *eval* command, Javascript has the ability to easily add code (in Javascript) that an execution engine runs as part of the main program. There is no need for libraries and a complicated plug-in systems, the new code is integrated in the existing one right away. Figure 5.1 depicts the impressive number of hardware and software environments which support HTML5 and as a result MuffinInfo.

A major advantage of Javascript (like Python and Perl) is the availability of the source code for the user without the need of using additional tools for building the application. This way, the code can be easily modified when the need arises and the modifications can be tested right away.

Finally, even though there are alternatives like FastQC, NGS QC Toolkit [213] and PRINSEQ [214] (none in HTML5), they have caveats as exemplified in the continuation of the paragraph. Table 5.1 list the most important features for a statistics extraction tool for the aforementioned programs and MuffinInfo. FastQC is the most complete of all but its Java implementation does not offer the same grade of freedom as HTML5 in addition to inability to reload results of a run in the same interactive UI and user-modifiable parameters. FASTX-Toolkit is command line only, in Perl and targets just Illumina and Roche 454. PRINSEQ requires Perl for local execution and the existence of a web server for remote

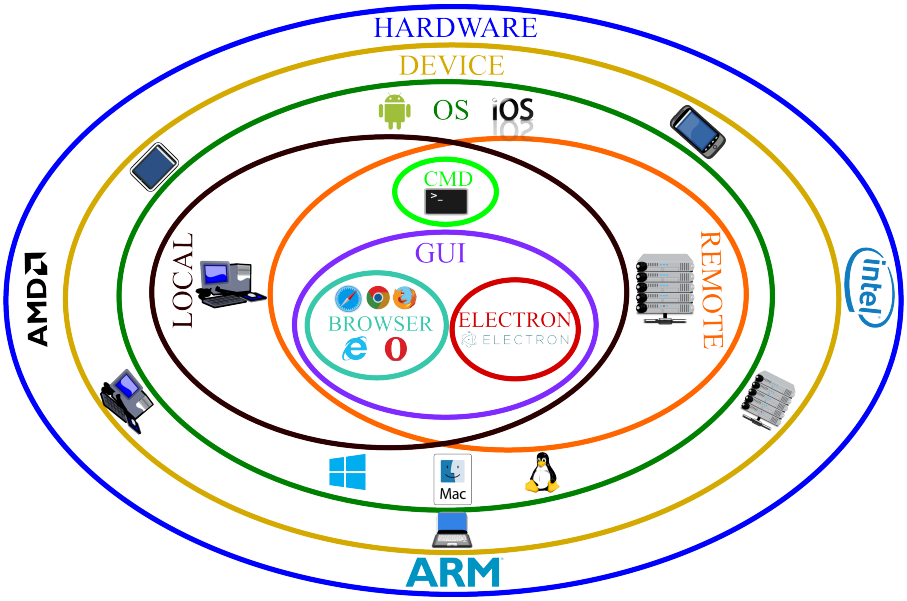


Figure 5.1: MuffinInfo can run in a multitude of environments. All logos appearing in this picture are the property of their respective owners. (Online version in colour.)

execution. Its standalone version doesn't work on Windows. None of the above support custom, user added statistics. The last two listed tools are more than information extractors, because they can also alter the data.

Table 5.1: Comparison with other similar software

Feature	**MuffinInfo**	**FastQC**	**NGS QC Toolkit**	**PRINSEQ**
Technology supported	Any	Any	Illumina/454	Any
Base/Color space	Y/N	Y/Y	Y/N	Y/N
Number of features	21	19	6	10
Features	Read count/min/max/with N, Num bases total/A/C/G/T/N, Qual profile, Qual per base, K-mer spectrum, % GC, Reads len list, Duplicates, Adapters, Homopolymers Len Distrib	File type, Enc, Read count, Filtered seq, Seq len, % GC, Per base/seq qual, Seq A/C/G/T, Per seq GC, Per base N, Seqs len, Duplicates, Overrepr seqs, Adapters, K-mer pos, Per tile seq qual	Avg qual, % GC, Reads per avg phred qual, Read count per base qual ranges, Base composition, Summary qual check/filtering	Reads len, Base quals, GC, Poly-A/T tails, Ambiguous bases, Duplicates, Complexity, Tag seqs, Contamination, Asm
GUI	Y	Y	N	Y
Command line	Y	Y	Y	N
Online	Y [1]	N	N	Y[2]
Offline	Y	Y	Y	Y
Custom statistics	Y	N	N	N
OS Support	Any[3]	Desktop	Desktop	Any[4]
Built in	HTML5	Java	Perl	Perl+HTML+PHP
Statistics settings	Y	N	N	N
Export results	JSON	HTML	HTML	HTML
Reopen / reuse old results	Y	N	N	N

[1] It requires just a static HTML server
[2] It needs an Apache server with Perl CGI for hosting
[3] As long as a HTML5 engine exists on a specific platform
[4] Network access required for the online version

Methods

MuffiInfo is an extensible, efficient, portable and useful NGS information extraction tool. Furthermore, by developing it we want to demonstrate the suitability of web technologies as a viable approach for bioinformatics applications. Server-side software solutions are common, but require costly communication transfers and the maintenance of non-trivial computing and storage back-ends. With the new features introduced in HTML5, it is now possible to write applications that were only able to be developed in traditional languages like Java and C++. The performance of HTML5 is sometimes very close to native code, as reported in [215] where a resource intensive video game engine was ported to HTML5. Last but not least, the latest Javascript standard enhances the performance by introducing optimized data structures (like efficient maps to hold key→value pairs instead of the all-purpose, traditional *Object* - the base class in Javascript).

Our software can run both online, served by a basic server (able to serve HTML pages) or offline from the local disk. Please note that MuffinInfo is executed on user's machine, not on the server. Therefore, once downloaded, MuffinInfo should work on a disconnected computer because all its files can be stored by the browser in the cache. This approach promotes the privacy of the data since no information is sent to a third party server. The user can also download the whole software and open locally *index.html*.

Even though there is no reliance on a server, we fully understand the need of one in some cases. As a result, we designed MuffinInfo to run both in a browser and in command line (by using Node.js [216]) where no GUI is available. The latter mode is useful for building higher-level scripts or pipelines that include an information extraction step. For example, a plugin for Galaxy [217] can be easily implemented. Furthermore, one can easily connect to

a remote machine with no GUI, perform the statistics extraction there and then return the results which can be displayed on a local machine. This scenario is likely to arise whenever the local machine is not powerful enough to process a large dataset, therefore a server (which usually comes without a GUI) can do the heavy lifting.

Our application revolves around the File API which allows us to read local files in chunks instead of being forced to read the data from a server. This way we avoid filling the RAM with the input data, sparing this fast memory for storing the statistics. Secondly, Web Workers (HTML5 threads) can speed up the computation and avoid blocking the UI by making use of the multi-core CPUs (now available even in phones and tablets). MuffinInfo, in its GUI form, uses two threads: one that handles the UI and the other that generates the statistics. In its command line form, MuffinInfo uses just one thread at the moment.

MuffinInfo include a number of 8 statistics presented in three forms (list of features, charts and tables):

- General information (like total numbers of nucleotides, reads and different types of bases)
- Quality Distribution per base
- K-mers histogram
- GC percentage per reads
- List of reads length
- List of duplicates
- List of adapters
- Homopolymers length distribution

A detailed description of the implemented characteristics and program behaviour are located in the Help→Contents menu of the application. We extract information from FASTA, FASTQ and SAM files. The extractor supports input files without quality scores, but the information presented is reduced.

MuffinInfo is extensible and provides a way to inject external code to implement custom statistics not yet supported by the tool, without having to explicitly recompile, re-install or even restart the software. In order to support this, we divided the execution of a statistic in four levels: initialization, loop read processing, statistics wrap up and display of the result as list of features, table or chart. This way, the user can control the exact behaviour of a statistic divided into the four levels. A more detailed explanation can be found in section 5.2.

Users might need to export the results for their inclusion in reports or other applications. Many tools only provide an HTML file with bitmap charts. This prevents further processing or automatic comparison. MuffinInfo stores the results of a run in JSON format (opening the door to further processing by third party applications). It can reopen a run and repopulate the tabs with the results. Our application uses HTML, therefore the application itself is the report.

MuffinInfo allows the user to select a number of parameters used in the statistics computation like the k-mer size. Once the values are set, our software will store them in the *localStorage* of the browser.

The UI presented in Figure 5.2 is in line with modern application design, resembling a desktop application. Therefore, there is a base window with a number of tabs, display area and a main menu which are not separated HTML documents. Instead of multiple HTML files connected together using links a a wizard, MuffinInfo relies on Javascript to modify the existing main document. For instance, when someone selects a statistics, MuffinInfo hides the

Figure 5.2: MuffinInfo main UI

current one and loads the selected one from the main memory. This way we can offer a desktop like feeling for our application and avoid forcing our user to use the back button.

Extensibility

MuffinInfo accepts customs statistics built by the user. The accepted code is Javascript, therefore the user inherits all the freedom of the language. Furthermore, a power user who knows Javascript can access all the program's hidden features by editing the code on the fly. This way the application can be easily modified by accessing different globally available objects like a certain chart (to modify its type for instance). Although we deliberately allow this, we do not recommended this approach. The reason for this is to avoid breaking the custom, user-side modified code with our new versions of the software. We try to expose as much of the internal workings of MuffinInfo as we can and find useful for adding a new statistic. This is like a public API that won't get modified with a minor release. The rest of this section explains how to add a custom statistic. One should first know that there is a publicly available *parser* object which holds the temporary data which is being updated at the second step as the data stream gets processed. When the processing of the file is complete, another object called *statistics* is made available. It is the link between the parser and the rest of the program, therefore the user must transfer the temporary held results from *parser* to *statistics*.

The first step deals with the declaration and initialization of all the custom properties of *parser*. At this first step, the user can access *parser* with all its properties and the settings (from the Settings window). The parser is the main engine of our software because it receives a chunk of text read from the disk and it extracts the entries (id + read + quality) from it.

The loop entry processing (the second step) represents the method called each time MuffinInfo extract an entry from the input file. Normally, an user would update the statistics at this step without declaring any new properties for the parser object. The custom

code has access to three additional new objects, namely the components of an entry. Please bear in mind that the quality object can be null (as it happens for fasta files where there are no quality scores).

At the third step, the parser invokes the finalization code when the stream reader has reached the end of the input file and all entries have been extracted and processed. The *statistics* object becomes available. This object already contains all the predefined statistics like the number of bases. The user is now required to transfer his/her custom implemented statistics attached to the parser object to the *statistics* object.

The last step moves away from the parser object into the display mechanism where the user can display the custom statistic using three methods: a list of fields, a table or a chart (they are mutually exclusive). The list of filed involves a predeclared array called "list". One can push fields declared as arrays with two elements. The first element represents the name of the field, while the second is the value. When the user wishes to generate a table, (s)he must use the table object containing two properties: "columnNames" which defines an array with all the desired columns and "datatablesInit" which is an object used to initialize a Datatables object. The format of the latter can be obtained from the Datatables manual. We pass the definition created by the user directly to the Datatables constructor. This way we offer full manoeuvre space for anyone wishing to use this type of display method.

Finally, one can create charts using the HighCharts library. We offer a charts object which can be modified by the user at will using the library's documentation. Due to the inner workings of MuffinInfo, we strongly advise the user against modifying the "chart.renderTo", "chart.width" and "chart.height" options. MuffinInfo automatically takes care of the layout and arrangement of the chart. The custom statistics are saved in the local storage of the browser.

Our program contains a template with a read count statistic as an example. In case of no already defined custom method, the extractor will automatically load the template. Please keep in mind that "Save" won't add the statistic to the execution pipeline. One must use the "Add" button which will also save the statistic. We also offer the possibility of prior validation of the introduced code before adding the statistic to the execution pipeline. This way we hope to avoid crashing the program during the overall the execution when the user might have waited a lot of time for the completion of the process only to be forced to start again.

Results

Table 5.2: The performance of MuffinInfo; Time in minutes; Datasets from *E. coli* K-12 substr. MG1655

Dataset	Tech	# Bases (Mb)	Time Laptop	Time Tablet	Time Phone
SRR350073_1	Illumina	149	1.53	10.78	4.90
SRR000868	Roche 454	59	0.50	3.25	1.20
ERR039477	Ion Torrent	36	0.31	2.10	0.81
SRR387257	PacBio	31	0.25	1.50	0.56
ERR764952_1	Oxford Nanopore	48	0.36	2.10	0.85

MuffinInfo extends and complements the capabilities offered by FastQC. Even though FastQC is written in Java, it cannot run on as many platforms as MuffinInfo (for instance Android and iOS). Actually, the probability of porting an HTML5 rendering engine (like Gecko-Firefox or WebKit-Chrome) to a new platform is much higher than porting the Java virtual machine (considering the expanding availability of the internet).

In contrast to the fast increase of computational power of various devices, the size of the output of a sequencing run is limited by the size of the real genomes. Additionally, the improvement of

accuracy of sequencing will reduce the need of increasing the coverage As a result, the sequencing files for the same species should normally not increase in size by orders of magnitude given the same sequencing technology. In conclusion, even though the computational resources limited devices (like phones) may not be able to process large datasets using web technologies at the moments, they should be able to do so in the very near future. Additionally, the sequencing coverage tends to have upper limits as shown in [218]. This means that after a certain coverage there is no real gain in the result and the money is wasted. There already are phones with 4 GB of RAM and 8 CPU cores. We intentionally selected three mobile devices, an Intel Core i7-2630QM laptop with 16 GB RAM running Debian 8 x64, a Samsung S6 phone with 3 GB RAM and an ARM Exynos 7420 octa-core CPU with Android 5.1.1 and an Asus Fonepad 7 ME372CG tablet with an Intel Atom Z2560 CPU and just 1 GB of RAM running Android 5.0. MuffinInfo obtained the results presented in Table 5.2 on different operating system, devices types and hardware platforms. The tests were performed using Google Chrome for Linux (laptop) and Android (tablet and phone) respectively.

Conclusion

MuffinInfo presents information such as average length, base distribution, quality scores distribution, k-mers histogram and homopolymers analysis. It improves upon the existing extractors by adding the ability to save and then reload the results obtained after a run as a navigable file (also supporting saving pictures of the graphs), by supporting custom statistics implemented by the user and by offering user-adjustable parameters involved in the processing.

MuffinInfo comes as a proof that nowadays it is possible to run

bioinformatics software on a wide range of environments by using plain HTML5. As a result, MuffinInfo can be executed:

- online, served from a server or offline from the local disk
- with a graphical user interface or command line on an execution environment like Node.js
- on any operating system supporting a HTML5 browser (Windows, Linux, Mac OS X, Android, iOS, Windows Mobile, Blackberry OS etc.)
- on any hardware platform (HTML5 is a high level language, it is dependent upon the underlying execution platform which is hardware specific)
- in a browser/execution environment or as a standalone (thanks to project Electron)

Our software is being updated on a regular basis. There are many more statistics that can be added by the users and we plan to stimulate the sharing of new 3rd party statistics in the future. MuffinInfo is aimed at becoming a first-step tool for many different projects using Next/Third Generation Sequencing data like variant calling, error correction, genome assembly etc. The main goals for future releases are: performance improvement, better support for adding custom statistics, more default statistics, improved mobile GUI for small screens and multiple statistics under the same tab.

CHAPTER 6

Conclusion

NGS has become an important technology, with a great range of application from cancer detection to stem cell analysis. It replaced the microarrays as the favourite technology for deciphering the DNA and RNA code. In order to tackle different biological problems, bioinformatics tools were developed for various problems. From the old Smith-Waterman alignment algorithm to genome assemblers (like Velvet and Newbler), aligners (BWA and Bowtie2), error correctors (Coral and Blue), information extractors (FastQC and NGS QC Toolkit), genome and alignment visualizers (IGV[232] and Artemis[233]), etc., these all are proofs of the impact NGS has on science.

This thesis has three main objectives which are built one on the top of the other. We discuss each one of them in the next three paragraphs.

As demonstrated in chapter 2, there is a lot of interest in error correction. We found 50 methods after an exhaustive search for the available software in the literature. This study helps us understand the domain, as we are also putting an accent on the actual sequencing technology. It shines some light over the weak points that remain to be tackled. Because of its target (the principles of sequencing) and its relation to error correction (normally the first step and the correctors are evaluated using assemblers), the review from chapter 2 represents the theoretical introduction and foundation for the whole thesis. The necessity of a review in the field is reinforced by many requests and reads of our paper just one week after publication. We conclude that there is room for further improvement especially on the biological aspects of the correction. Here, we refer to concepts from biology like ploidy, heterozygosity and repetitive regions not the more computer science oriented concepts like the memory consumption, genome representation on two bits per base and multi-core support. Now that the error correction field has been sufficiently explored, we are at a level where the newer methods improve over the existing ones. A not so favourable

trend is the non-existence of mature methods that are constantly enriched with new features, as in other related fields like assembly (Mira [234]) or short sequences alignment (BWA [204]).

Usually, the information known "a priori" about a newly sequenced organism is limited. Even re-sequencing the same organism can result in an unpredictable output. In this thesis, we introduce MuffinInfo, a FastQ/Fasta/SAM information extractor implemented in HTML5 capable of offering insights into NGS data. It presents information such as average length, base distribution, quality scores distribution, k-mers histogram and homopolymers analysis. MuffinInfo improves upon the existing extractors by adding the ability to save and then reload the results obtained after a run as a navigable file (also supporting saving pictures of the graphs), by supporting custom statistics implemented by the user and by offering user-adjustable parameters involved in the processing. At the moment, the extractor works with all base space technologies such as Illumina, Roche, Ion Torrent, Pacific Biosciences and Oxford Nanopore. With the use of HTML5, our software demonstrates the readiness of web technologies for mild-intensive tasks encountered in bioinformatics. The utility of MuffinInfo is demonstrated by its acceptance at a prestigious conference (ISMB / ECCB) and then its publication in a first quarter‘ journal (Journal of Computational Biology).

Error correction is typically the first step of any application targeting NGS data. The majority of available stand-alone error correction solutions can only detect and correct mismatches. Therefore, these solutions only support correcting reads generated by Illumina sequencers. Several solutions support insertions and deletions (indels) and are capable of working with multiple technologies. However, they are limited by correction performance and resources consumption. This thesis presents MuffinEC, an indel-aware multi-technology correction method for NGS data. It uses a greedy approach to create groups of reads and subsequently cor-

rects them using their consensus. MuffinEC surpasses existing solutions by offering better correction ratios for multiple technologies. We tested it with different approaches: percentage of corrected errors in the form of gain specificity and sensitivity, percentage of short alignment after correction compared with the percentage before, assembly metrics, unknown bases count before and after correction. MuffinEC wins in all tests against similar, current software. The error correction method also exploits parallel processing via OpenMP and uses less computational resources than similar programs, thereby being capable of handling larger datasets on the same hardware setup. Additionally, we cover the evolution of sequencing technologies too by tackling all types of errors and allowing input from multiple sequencing technologies.

As the sequencing market share suggests, Illumina has become an important player in the industry. Our review strongly supports this claim based upon the general support for this technology, with almost all programs supporting either only Illumina or Illumina plus additional sequencing technologies. Pacific Biosciences and Oxford Nanopore technologies with their (very) long reads gave birth to a new trend. This trend requires the evolution of error correction techniques to support longer reads and to deal with the high error rate these technologies currently have. Overall we can see improvements in the area of error correction for different technologies as the newest methods are both resource efficient and offer a very good correction. A reliable and structured way to measure the accuracy is also very important.

Published results

This thesis resulted in multiple publications in high impact journals and at conferences:

1. Alic AS, Tomas A, Medina I, Blanquer I. *MuffinEc: Error correction for de Novo assembly via greedy partitioning and sequence alignment.* Information Sciences (JCR 2014 COMPUTER SCIENCE, INFORMATION SYSTEMS 6/139; Impact factor: 4.038). 2016 Feb 1;329:206-19.
   DOI: 10.1016/j.ins.2015.09.012

2. Alic AS, Ruzafa D, Dopazo J, Blanquer I. *Objective review of de novo stand-alone error correction methods for NGS data.* 2016 Jan 11; WIREs Comput Mol Sci (JCR 2014 MATHEMATICAL & COMPUTATIONAL BIOLOGY 1/57; JCR 2014 CHEMISTRY, MULTIDISCIPLINARY 11/157; Impact factor: 11.885). DOI: 10.1002/wcms.1239

3. Alic AS, Blanquer I. *MuffinInfo: HTML5-based statistics extractor from Next Generation Sequencing data.* Journal of Computational Biology (In Press). 2016

4. Alic A, Blanquer I. *MuffinInfo: HTML5 statistics extraction system from FastQ/Fasta/Sam files.* 23rd Annual International Conference on Intelligent Systems for Molecular Biology and the 14th European Conference on Computational Biology. 2015. DOI: 10.7490/f1000research.1110049.1

5. Alic A, Tomás A, Salavert J, Medina I, Blanquer I. *Robust Error Correction for De Novo Assembly via Spectral Partitioning and Sequence Alignment.* 2nd International Work-Conference on Bioinformatics and Biomedical Engineering (IWBBIO). 2014. 1040-1048

Software

We are strong supporters of open source software. Our work is available free of charge, for academic and industry use, at the following locations:

1. MuffinEC: `http://sourceforge.net/projects/muffinec/`
2. MuffinInfo: `http://sourceforge.net/projects/muffininfo/`

APPENDIX A

Error correction in real projects

The software presented so far was used in many research projects. This section is the extension of the motivation, where we present instances where scientist successfully employed error correctors. Hopefully, this section opens the door for further usage in comparable or new cases. We concerted our efforts to find real life biological projects where the utility of correctors is demonstrated by practical use. In contradiction to what we found in the articles accompanying each corrector, some real life projects use the correctors for additional applications like mithocondrial genome correction[139] and RNA-seq[140, 141, 142, 143]

Table A.1: Work using the correctors included in the current review

Year	Study Description	Where
	Quake/Illumina	
2012	Exomes comparison of *C. carpio* & *D. rerio*	[144]
2012	GAGE, evaluation of genome assemblies/algorithms	[78]
2013	Salinarchaeum HArcht-Bsk1T genome	[145]
2013	*L. arenae* draft genome	[146]
2013	*Citrus sinensis* draft genome	[147]
2013	Genetic variants in *C. sinensis*	[148]
2013	Genome-wide mutations in diploid yeast	[149]
2014	Results of correction of heterozygous NGS	[150]
2014	*O. sativa* de novo assemblies, novel gene space aus/indica	[151]
2014	Study of hydrocarbons production from fatty acids in Cyanobacteria	[152]

2014	Genetic diversity in *P. pacificus* from population-scale resequencing	[153]
2014	Genetic parameters estimation and response to selection in breeding program of M. Galloprovincialis	[154]
2014	Metagenomic characterization of *C. defluviicoccus* tetraformis	[155]
2014	Prediction of antibiotic resistance by gene expression profiles	[156]
2014	Methicillin resistance in *S. aureus*	[157]
2014	De novo creation of repeat libraries from whole-genome NGS reads	[158]
2014	Aerobic fungal degradation of cellulose	[159]
2014	*B. tryoni* draft genome	[160]
2014	Genome reorganization	[161]
2015	*P. vulgata*/*P. lamarcki* draft genomes	[162]
2015	The domestic dromedary genome	[163]
2015	The brown kiwi genome	[164]
2015	Comparative Genomics of *S. pyogenes* M1	[165]
2015	Approach for Identification and Characterization of Foodborne Pathogens	[166]
2015	*P. glaucus* complete mitochondrial genome	[139]
2015	Mechanisms for Speciation and Caterpillar Chemical Defense	[167]

BayesHammer/Illumina

Year	Study Description	Where
2014	GABenchToB, assembly benchmark for bacteria genomes	[168]
2014	*C. burnetii* genome	[169]
2014	*P. atrosepticum* genome	[170]

2014	Hidden diversity in honey bee gut symbionts	[171]
2014	*S. lemnae* draft genome	[172]
2015	Discovering Natural Products from Cyanobacteria	[173]
2015	Characterize the metabolism of *M. thiooxydans* L4 in the marine environment	[174]
2015	Utilization of alginate and other algal polysaccharides by marine *Alteromonas macleodii* ecotypes	[175]
2015	Genome-Wide Re-distribution in Active Yeast Genes	[176]
2015	Study of the metabolome of *M. producens* JHB	[177]
	Reptile/Illumina	
2014	Decrypting cryptobiosis-analyzing anhydrobiosis using transcriptome sequencing	[140]
2015	SNP genotyping and population genomics from expressed sequences	[178]
	HSHREC/Illumina,454	
2014	Decrypting cryptobiosis-analyzing anhydrobiosis using transcriptome sequencing	[140]
	BLESS/Illumina	
2014	Transcriptome, sequence polymorphism, and natural selection in *P. eremicus*	[141]

	Blue/Illumina	
2014	*S. scitamineum* genome	[179]
	Coral/Ion Torrent[(a)], Illumina[(b)], 454[(c)]	
2014	GABenchToB, assembly benchmark for bacteria genomes[(a)]	[168]
2014	Global gene expression in the exocarp of developing *P. avium* L.[(b)]	[142]
2015	Comparative genomics/gene expression applied on *P. xuthus* and *P. machaon* genomes[(c)]	[180]
	DecGPU/Illumina	
2013	Genomic analysis of *S. dulcamara*	[181]
	Echo/Illumina	
2012	Pipeline for small RNA-seq data analysis	[143]
2014	Results of correction of heterozygous NGS	[150]
2014	Assembly/annotation for *T. pratense*	[182]
	Freclu/Illumina	
2011	MicroRNA-mediated gene regulation role	[183]
2011	Purification of monocyte subsets from *H. sapiens* blood and their transcriptomes analysis	[184]
2013	Identification of functional cis-regulatory elements	[185]
	Hector/454	

2015	Triple-negative breast cancers in patients with no BRCA1 or BRCA2 mutation	[186]
	Lordec/PacBio,Illumina	
2015	De novo tandem repeat detection using short&long reads	[187]
	LSC/PacBio,Illumina	
2014	*D. officinale* genome and genes analysis	[188]
2015	Detect fusion genes, determine fusion sites and identify and quantify fusion isoforms	[189]
2015	*S. miltiorrhiza* transcriptome and tanshinone biosynthesis insights	[189]
	proovread/PacBio,Illumina	
2015	Characterization of venom toxin-encoding genes in *E. coloratus*	[190]
	QuorUM/Illumina	
2014	*P. taeda* reference genome	[3]
	RECOUNT/Illumina	
2012	Brain tumor glioblastoma-derived neural stem cells transcriptome analysis	[191]

Recommendations

Quake[9] is the most used corrector as it targets the sequencing technology with the largest market share. Furthermore, as we

shall see in section "Testing", it has a good level of correction. As a general rule, Illumina data should be handled by Illumina-only correctors since they should be better tuned for the technology than their general counterparts. There are exceptions like **Coral**[15] that is used for both Illumina and 454. The same software is utilized with Ion Torrent for which there is a generic support. Generally, the software supporting all types of errors can be used with unsupported technologies, but the user must understand that the result might not be what expected. In the above case, Ion Torrent is somewhat similar to 454, hence **Coral** works. In the case of datasets from multiple technologies, one can use more than one corrector for each technology as Wang *et al.* did[140]. Another possibility is to use a cross-corrector like **Blue**, **LorDEC**, **proovread** and **LSC** where instead of stacking up all the reads from multiple technologies, one can use one technology to correct the other. The correctors are used in many types of projects with the obviously most targeted being assembly. Variant calling and different transcriptome studies are also very common in the existing projects. A very important fact emerging from table A.1 is the range of genomes data tackled with the stand-alone correctors. The size and complexity ranges from bacteria (*S. pyogenes*) to mammals (Dromedary) and plants (Loblolly Pine).

APPENDIX B

External Testing of Error Correctors

Table B.1: Testing result from benchmarks and original papers; **App** - the name of the corrector, **Mem/Rt/Gain** - memory in GB/runtime in mins (num of threads in parentheses for **Rt O**)/gain as percentage reported in the original paper (represented by **O**) and in the survey by [1] (represented by **S**), **Gain R** - gain(percentage) reported by [2], **# Rds** - number of reads, **# Bs** - number of bases; The programs that didn't run successfully are market by "-*".

Dataset	Genome	Tech	# Rds	# Bs	App	Mem O	Mem S	Rt O	Rt S	Gain O	Gain S	Gain R
					SHREC	-	-	-	-	-	-	0.75
					HSHREC	-	14.3	-	153.23	-	-2.886	0.989
					HiTEC	-	13	-	143.13	0.861	0.846	0.952
					DecGPU	-	-	-	-	-	-	0.772
					Coral	-	7.5	-	36.19	-	0.579	0.857
					ECHO	-	-*	-	-*	-	-*	0.715
					Hammer	-	-	-	-	-	-	0.944
SRX000429	*E. coli* 36	Illum	10.4	749.4	Quake	-	4.1	-	38.88	-	0.784	-*
					REDEEM(i)	9	-	120(1)	-	-	-	0.926
					RACER(d)	1.1	-	15.7(1)	-	0.908	-	-
					Trowel	3.5	-	2.9(32)	-	0.765	-	-
					CloudRS	-	-	-	-	0.207	-	-
					Reptile(e)	1.1	3.7	47.4d 149.4d2(1)	23.32	0.757d1 0.802d2	0.933	-0.047
					Par. Reptile(e)	-	-	0.18d1 1.38d2(512)	-	0.757d1 0.802d2	-	-
					SHREC	-	-	-	-	-	-	0.822
					HSHREC	-	-	-	-	-	-	0.989
					HiTEC	-	-	-	-	0.857	-	0.795
					DecGPU(b)	-	-	12.31(4)	-	-	-	0.999
SRR001665_1(j)	*E. coli* K12 MG1655	Illum.	10.4	749.4								

Table B.1 Continued from previous page

Dataset	Genome	Tech	# Rds	# Bs	App	Mem O	Mem S	Rt O	Rt S	Gain O	Gain S	Gain R
					Coral	-	-	-	-	-	-	0.999
					ECHO	-	-	-	-	-	-	0.999
					Hammer	-	-	-	-	-	-	0.668
					REDEEM	-	-	-	-	-	-	0.231
					BLESS	0.01	-	23(1)	-	0.967	-	-
					Qamar(f)	3.5	-	38.85(-)	-	0.913	-	-
					Reptile(e)	0.84	-	21d1 73.8d2(1)	-	0.652d1 0.709d2	-	0.998
					FADE	-	-	0.51(FPGA)	-	0.923	-	-
					SHREC	-	-	-	-	-	-	0.89
					HiTEC	-	-	-	-	-	-	0.324
					DecGPU	-	-	-	-	-	-	0.548
					Coral	-	-	-	-	-	-	0.816
					ECHO	-	-	-	-	-	-	0.711
SRR006332	Acinetob. ADP1	Illum.	18.13	652.7	Hammer	-	-	-	-	-	-	0.984
					Reptile(e)	2.2	-	99.6d1(1)	-	0.632d1	-	0.982
					Quake	-	-	-	-	-	-	0.926
					CUDA-EC	-	-	-	-	-	-	0.953
					REDEEM	-	-	-	-	-	-	0.595
					CloudRS	-	-	-	-	0.265	-	-
					SHREC	-	-	-	-	-	-	0.997
					HSHREC	-	-	-	-	-	-	0.995
					HiTEC	-	-	-	-	-	-	0.994
					DecGPU	-	-	-	-	-	-	0.995
					Coral	-	-	-	-	-	-	0.997
NC_005966	Acinetob. ADP1	Illum.	4	144	ECHO	-	-	-	-	-	-	0.993
					Hammer	-	-	-	-	-	-	0.997
					Reptile(e)	0.66	-	15.6d1(1)	-	0.599d1	-	0.995
					Quake	-	-	-	-	-	-	0.993

Table B.1 Continued from previous page

Dataset	Genome	Tech	# Rds	# Bs	App	Mem O	Mem S	Rt O	Rt S	Gain O	Gain S	Gain R
					CUDA-EC	-	-	-	-	-	-	0.995
ERA000206	*E. coli* K12 MG1655	Illum.	14.21	2800	HSHREC	-	29.9	-	779.15	-	-1.606	-
					Reptile	-	19.1	-	225.43	-	0.91	-
					HiTEC	-	9.8	-	683.87	0.874	0.949	-
					Coral	-	30	-	450.29	-	0.112	-
					Blue(a)	1.6+ 0.4	-	9.2+ 14.4(8)	-	-	-	-
SRR022918	*E. coli* 47	Illum.	7.2	677.2	HSHREC	-	12.6	-	74.13	-	-0.316B -0.269R	-
					Reptile(e)	1.9	3.4	56.4d1(1)	71.64	0.381d1	0.852B 0.791R	-
					Quake	-	2	-	80.43	-	0.313B 0.285R	-
					HiTEC	-	6.2	-	59.19	-	0.929B 0.912R	-
					ECHO	-	16	-	304.21	-	0.907B 0.897R	-
					Coral	2.6	7.7	8.12(4)	8.32	0.974	0.002B 0.002R	-
					RACER(d)	1.5	-	13.65(1)	-	0.826	-	-
					Trowel	4.5	-	3.7(32)	-	0.447	-	-
SRR000868	*E. coli* UTI89	454	0.4	118.2	Fiona	1	-	2.5(8)	-	0.768	-	-
					Hector(c)	-	-	3(2)	-	0.882	-	-
					HSHREC(h)	-	9.9	-	16.72	-	0.293E1 0.224E2 0.470E3 0.473E4 0.451E5	-
					Coral(h)	1.8	2.4	5.3(4)	2.57	0.857	0.497E1 0.425E2 0.698E3 0.702E4 0.734E5	-

Table B.1 Continued from previous page

Dataset	Genome	Tech	# Rds	# Bs	App	Mem O	Mem S	Rt O	Rt S	Gain O	Gain S	Gain R
SRX100885	*S. cerevisae*	Illum.	26.03	4000	HSHREC	-	30.1	-	1027.48	-	-2.497	-
					Reptile	-	4.12	-	165.88	-	0.224	-
					Coral	-	34.5	-	544.31	-	0.067	-
					RACER(d)	3.2	-	198.31(1)	-	0.146	-	-
					CloudRS	-	-	-	-	0.779	-	-
					Chung *et al.*	-	-	153.48(80)	-	-	-	-
SRR022866	*S. aureus* USA300	Illum.	12.77	1900	HSHREC	-	30.2	-	813.65	-	-4.24	-
					Reptile	-	13.1	-	295.94	-	0.613	-
					Coral	4	40	80(4)	210.17	0.974	0.025	-
					RACER(d)	2.1	-	81.68(1)	-	0.47	-	-
					Trowel	5.7	-	6.4(32)	-	0.447	-	-
					CloudRS	-	-	-	-	0.335	-	-
					Chung *et al.*	-	-	63.45(80)	-	-	-	-
SRX023452 SRX006152 SRX006151	*D. melanogaster*	Illum.	18.96 9.46 20.8	3600 1400 1900	HSHREC	-	30.3	-	2562.88	-	-5.85	-
					Reptile	-	24	-	532.76	-	0.667	-
					Quake	-	12.1	-	222.35	-	-0.126	-
					Coral	-	30	-	373.19	-	0.449	-
					RACER(d)	39.4	-	501.95(1)	-	0.57	-	-
					Par. Reptile(e)	-	-	2.82d1 412.98d2(512)	-	-	-	-
ERR039477	*E. coli* DH10B	Ion	0.39	36	Fiona	1	-	3.1(8)	-	0.905	-	-
					HSHREC(g)	-	10.21	-	8.72	-	0.399T1 0.450T2	-
					Coral(g)	-	2.24	-	1.13	-	0.515T1 0.604T2	-

Table B.1 Continued from previous page

Dataset	Genome	Tech	# Rds	# Bs	App	Mem O	Mem S	Rt O	Rt S	Gain O	Gain S	Gain R

(a) Resources for preparing the dataset and correcting the data;
(b) The result for 1xGPU execution, for more threads and GPUs please refer to [103];
(c) Approximate values for runtime extracted from the chart in the paper for the original results;
(d) Results for the datasets filtered using BWA;
(e) Runtime for Hamming dist (1 and 2) between two k-mers;
(f) Gain calculated by us using the values for TP, FP, FN given by the authors and the formula in [1];
(g) Profiles: T1 - With all mapped reads; T2 - excluding reads with more than 10 errors;
(h) Different profiles for the aligner and a k-mer length of 10; The parameters set for E(1-5) can be found in [1];
(i) Dataset not explicitly pinpointed, we inferred the value;
(j) SRR001665_1 is the same as SRX000429, but only the forward direction is used;

APPENDIX C

Testing Methods for Correctors

Table C.1: Testing methods algorithms; RC represents the resource consumption

Name	**Test Real Data**	**Test Sim Data**	**Type Test Genomes**	**Programs Tested Against**	**Method of Testing**	**RC**
CUDA-EC	y	y	*S. cervisae, E. coli, H. influenzae, S. aureus, H. acinonychis*	SAP (EULER-SR)	Specificity, Sensitivity, 1-2 error / read Corrected / Trimmed / Accuracy	y
Reptile	y	n	*E. coli*, Acinetobacter ADP1	SHREC	Specificity, Sensitivity, Gain	y
Quake	y	y	*E. coli, H. sapiens, M. rotundata*	SoapdeNovo, SHREC, Euler	Accuracy, Assembly(Velvet, Soapdenovo), SNP calling	y
EDAR	y	y	Human bac, *P. gingivalis, S. aureus*	SAP (Euler-SR)	Error rate, Assembly(Velvet)	n
Hammer	y	n	*E. coli*	Quake, HiTEC	Positive Predictive Value(k-mers), Sensitivity (k-mers), Cluster of k-mers	y
REDEEM	y	y	*N. meningitidis*, Maize, *E. coli*	SHREC, Reptile	Specificity, Sensitivity, Gain, true err dist (tIED), wrong Ill err dist (wIED), true uniform err dist (tUED), wrong uniform err dist (wUED	y
DecGPU	y	y	*E. coli, D. melanogaster, S. aureus, M. agalactiae*	hSHREC	Sensitivity, specificity, By-base error rate, correct correction rate, incorrect correction rate, and the rate of newly introduced errors, Assembly (Velvet, Abyss)	y
CUDA-EC2	y	n	*M. agalactiae, S. aureus, E. coli*	Euler-SR, CUDA-EC	FP/FN for trimming, Assembly (Edena)	y

Table C.1 Continued from previous page

Name	Test Real Data	Test Sim Data	Type Test Genomes	Programs Tested Against	Method of Testing	RC
Qamar	y	y	*D. melanogaster, B. vulgaris, Acinetobacter ADP1, H. acinonychis, S. aureus, E. coli, S. cerevisiae, H. influenzae, B. suis, S. oneidensis, S. epidermidis, Wolbachia sp, C. burnetii, C. caviae, C. jejuni, E. litoralis, G. coronavirus*	HiTEC	Accuracy, Specificity, Sensitivity, Mapped/Unmapped, Identity	y
Parallel Reptile	y	n	*E. coli, D. melanogaster*	Reptile	None (Only perf)	y
BayesHammer	y	n	*E. coli, S. aureus*	Camel, Euler-SR, Coral, Hammer, Quake	k-mers count, reads mapping perc, Assembly (SPAdes)	y
QuorUM	y	n	*R. sphareoides, S. aureus, M. musculus*	Coral, ECHO, HiTEC, Quake	Error corr chimeric reads, Percent of false 31-mers remaining and true 31-mers missing, Assembly (estimation), Percentage of the original reads that are perfect after correction and percentage of sequence contained in perfect reads, compared with the original reads	n
RACER	y	n	*L. lactis, T. pallidum, E. coli, B. subtilis, P. aeruginosa, L. interrogans, H. influenzae, S. aureus, S. cerevisiae, C. elegans, D. melanogaster*	Coral, HiTEC, Quake, Reptile, SHREC	gain	y

Table C.1 Continued from previous page

Name	Test Real Data	Test Sim Data	Type Test Genomes	Programs Tested Against	Method of Testing	RC
Musket	y	y	*E. coli, H. sapiens*	HiTEC, SGA, SHREC, Coral, Quake, Reptile, DecGPU	Recall, Precision, F-score, Gain, Assembly (SGA)	y
Hector	y	y	*E. coli, S. enterica*	Coral, Acacia	Recall, Specificity, Gain, Precision, F-Score	y
Lighter	y	y	*E. coli, H. sapiens*	Quake, Musket, BLESS	Recall, Precision, F-score, gain, assembly (velvet)	y
HErCoOl	y	n	Hepatitis C	Coral, KEC, Reptile	Error rates (k-mer variation), Error percentage (counting mismatches, indels in homopolymer vs non-homopolymer regions)	y
Trowel	y	y	*A. thaliana, S. aureus, S. cerevisae, D. melanogaster, H. sapiens*	Coral, Musket, SoapEc, Quake	Read Accuracy, Base Accuracy, Transcriptome data accuracy, SNP calling, Assembly(...)	y
LoRDEC	y	n	*E. coli*, Yeast, Parrot	PacBioToCA, LSC	Sensitivity, Gain, Alignment Percentage, Coverage (Expected vs Observed)	y
BLESS	y	y	*S. aureus, E. coli, H. sapiens*	Quake, Reptile, HiTEC, ECHO, Musket, DecGPU	Sensitivity, Specificity, Gain, Alignment (Bowtie), Assembly (Velvet, SOAPdeNOVO, SGA)	y
Blue	y	n	*E. coli, H. sapiens, P. aeruginosa*	BLESS, Coral, HiTEC, HSHREC, Racer, Reptile, SHREC	Reads Mapping Stat after Correction, Assembly Mauve Metrics (Velvet)	y
BFC	y	n	*H. sapiens, C. elegans*	BLESS, Bloocoo, Fermi2, Lighter, Musket, SGA	Reads count(Perfect, Chimeric, Better Worse), Assembly(Velvet, Abyss, fermikit), Potential FP SNP	y

Table C.1 Continued from previous page

Name	Test Real Data	Test Sim Data	Type Test Genomes	Programs Tested Against	Method of Testing	RC
Scrible	y	y	*O. sativa, H. vulgare*	SGA, RACER	Percent Mapping Reads (0-3 Mismatches), Assembly (Contigs count, N50; Velvet)	y
PAGANtec	n	n	NA	None	None	y
ACE	y	n	*E. coli, M. tuberculosis, S. enterica, L. monocytogenes, P. syringae, B. dentium, O. tsutsugamushi, L. pneumophila, C. elegans, D. melanogaster, S. sapiens*	SGA, RACER, HiTEC, BLESS	Depth/Breadth Read/K-mer Gain	y
FADE	y	y	*S. aureus, E. coli, H. sapiens*	BLESS, Lighter, Musket	Sensitivity, Specificity, Gain	y
Pollux	y	n	*S. aureus, E. coli, H. sapiens, L. pneumophila, M. tuberculosis, R. sphaeroides*	Quake, SGA, BLESS, Musket, Racer	Corrections, Reads removed (%), Errs. corrected(%), Errs. introduced (%), Assembly(Velvet)	y
Gu *et al.*	n	y	*E. coli*	None	Corrections count, Accuracy	y
Jabba	n	y	*N. meningitidis, A. hydrophila, D. melanogaster, E. coli*	LoRDEC	Specificity, Sensitivity, Gain, Precision	y
SHREC	y	y	*S. cerevisiae, H. influenzae, E. coli*	SAP (EULER-SR)	Specificity, Sensitivity, Accuracy, Assembly (Edena), Percent Corrected Reads	y
HSHREC	y	y	*E. coli*	SAP (EULER-SR), SAET	Specificity, Sensitivity, Accuracy, Error Rates (before/after corr), Assembly(Velvet)	y
PSAEC	n	y	*S. cerevisae, H. influenza, E. coli, S. aureus*	HiTEC, SHREC, Reptile	None (Only perf)	y

Table C.1 Continued from previous page

Name	Test Real Data	Test Sim Data	Type Test Genomes	Programs Tested Against	Method of Testing	RC
HiTEC	y	y	*S. aureus, E. coli, H. influenzae*	SHREC, reptile	Accuracy	y
MyHybrid	n	y	*S. cerevisiae, H. influenzae, S. aureus, E. coli*	HiTEC, SHREC, Quake, hshcrec	Accuracy	y
Pluribus	n	y	*H. sapiens*	Quake, HSHREC	Precission, Recall, Assembly(Velvet)	y
Fiona	y	n	*D. melanogaster, H. sapiens, E. coli, S. aureus, S. cerevisae, P. falciparum, B. pertussis*	Allpaths-LG, Coral, HSHREC	Error Rate, Gain	y
ECHO	y	y	PhiX174, *D. melanogaster, H. sapiens, S. cerevisiae*	SA (Euler-USR), SHREC	Gain, By-base error rate, by-read error rate, Assembly (Velvet)	y
Coral	y	y	*E. coli, S. aureus*	SHREC, Quake, Reptile	Sensitivity, Gain, Assembly (Edena)	y
LSC	y	n	*H. sapiens*	PacBioToCA	Average length of output sequences, Accuracy (map the corrected reads on the original genome), Sensitivity/Specificity exon junction detection	y
CloudRS	y	n	-	SHREC, Reptile, Coral, HSHREC, SOAPEc	Specificity, Sensitivity, Gain, Precision, Assembly (Velvet)	y
Chung *et al.*	y	n	*S. cerevisiae, S. aureus, E. coli*	CloudRS	Number of Votes	y
proovread	y	n	*E. coli, A. thaliana, H. sapiens*	PacBioToCA, LSC	Accuracy (map the corrected reads on the original genome), Assembly	y
Nanocorr	y	n	*E. coli*, Yeast	None	Alignment(BLAST), Assembly(Celera)	n

Table C.1 Continued from previous page

Name	Test Real Data	Test Sim Data	Type Test Genomes	Programs Tested Against	Method of Testing	RC
Karect	y	n	*H. pylori, Z. mobilis, E. coli, S. aureus, H. sapiens, C. elegans*	Lighter, Trowel, BLESS, Musket, RACER, SGA, Quake, Reptile, Diginorm, Blue, Fiona, DAG-Con, Coral, MuffinKmeans, HSHREC	Base-operations & Whole Reads: Recall, Precision, FScore, Gain, Assembly(Contigs & Scaffolds: NGA50, LGA50, GM, LM, UA, MM, Coverage)	y
FREClu	y	n	*D. melanogaster, H. sapiens*	-	Clustering (map representative sequence in a cluster)	y
[61]	y	y	PhiX, Wheat	-	Specificity, Sensitivity	n
RECOUNT	y	y	*B. vulgaris, D. melanogaster, E. coli, M. musculus*	frec	Num mapped reads, Changes expression of known genes, falsely mapped reads to the wrong genome	y
Premier	y	y	*E. coli*	Reptile	Probability error correction, gain, total number of ground truth errors in the sequencing reads excluding those in the first k-mer	n
Premier Turbo	y	y	*E. coli, C. elegans*	Reptile, Musket, HiTEC, SHREC, Quake	Num. Errs. Correctly Recovered, Sensitivity, Num. Errs. Falsely Introduced, Gain	n
kGEM	n	y	Hepatitis C	QuasiRecomb	Sensitivity, Positive Predicted Value	n

Concluded

APPENDIX D

Correctors' Performance

Table D.1: Algorithms' performance on different datasets

Dataset	Genome	Tech	# Reads Mil**	# Bases Mil**	Programs	Mem GB	Rt mins (Num Th)	Gain	N50*
									Concluded
SRR034509_1	*E. coli* K12 MG1655	Illum.	10.4	2100	Reptile	4.6	165.6(1)	0.789	-
					Par.	-	0.38d1	0.789d1	-
					Reptile(h)		19.86d2(512)	0.839d2	
					CloudRS	-	-	0.043	76845(V)
					Chung *et al.*	-	56.53(80)	-	59.8(A)(p)
NIDS2(o)	*S. aureus*	Illum.	3.8	138	CUDA-EC(d,l)	-	3.63er1 6.88er2(1)	0.995	-
					SHREC(l)		12.31(-)	0.96	-
					Coral	1.7	4.25(4)	0.937	-
					Qamar(l)	2	16.25(-)	0.977	-
					HiTEC	-	-	0.933	-
SRR006331	*M. agalactiae* PG2 MaPG	Illum.	1.7	61	DecGPU(c)	-	1.18(4)	-	7.4(V) 4.7(A)
					CUDA-EC2(j)	-	1.01(1)	-	2.9(E)
SRR016146	*S. aureus*	Illum.	4.4	226.6	DecGPU(c)	-	6.46(4)	-	34.8(V) 34.8(A)
					CUDA-EC2(j)	-	4.48(1)	-	32.5(E)
NIDS3(o)	*E. coli*	Illum.	2.6	91.7	HiTEC	-	-	0.904	-
					Qamar(l)	2.3	15.48(-)	0.731	-
SRR031259	*S. cerevisiae*	Illum.	7.5	269.5	ECHO	-	-	0.211	-
					Fiona	3.21	4.64(8)	0.588	-
NDIS12(o)	*H. acinonychis*	Illum.	8.2	295.2	CUDA-EC(d,l)	-	7.56er1 14.8er2(1)	0.997	-
					HiTEC	-	-	0.912	-

Table D.1 Continued from previous page

Dataset	Genome	Tech	# Reads Mil**	# Bases Mil**	Program	Mem GB	Rt mins (Num Th)	Gain	N50 (kb)*
					Qamar(l)	10.6	38.03(-)	0.503	-
ERR022075	*E. coli* K12 MG1655	Illum.	22.7	4600	Fiona	16.85	56.43(8)	0.986	-
					Lighter	-	-	-	98555(V)
					Trowel	8	9.9(32)	0.976	107.4(S) 113.7(V)
					Musket(g)	-	0.41(2)	0.975	13.2(Sg)
					Premier Turbo Fanop	-	-	0.979	-
					Premier Turbo A-Viterbi	-	-	0.982	-
SRR065390	*C. elegans*	Illum.	33.8	6800	RACER(f)	16.7	551.2(1)	0.659	-
					Musket	-	35.81(2)	-	9.2(Sg)
					Premier Turbo Fanop	-	-	0.892	-
					Premier Turbo A-Viterbi	-	-	0.894	-
					Karect	147.83	102.93	0.875	- 33700(V)
					BFC-bf	-	-	-	33.7(A)() 34800(V)
					BFC-ht	-	-	-	34.2(A)
SRR022868	*S. aureus* USA300	Illum.	15.56	3100	BLESS	0.01	6(1)	0.894	-
					Pollux	-	3.67(-)	-	1771(V)
					FADE	-	0.1(FPGA)	0.879	-
					Karect	2.58	3.25(12)	0.994	27.5(V) 24.6(Sg) 24.1(C)

Table D.1 Continued from previous page

Dataset	Genome	Tech	# Reads Mil**	# Bases Mil**	Program	Mem GB	Rt mins (Num Th)	Gain	N50 (kb)*
SRR016390	*S. aureus*	Illum.	10.3	525.3	CUDA-EC2(j)	-	6.16(1)	-	28.2(E)
SRR011186	*M. canettii*	Illum.	10.6	809.4	CUDA-EC2(j)	-	15.71(1)	-	8.6(E)
SRR016399	*S. aureus*	Illum.	12.7	965.7	CUDA-EC2(j)	-	21.41(1)	-	37.6(E)
SRR026446	*M. tuberculosis*	Illum.	20	3000	CUDA-EC2(j)	-	-	-	2.9(E)
ERR161541	*B. pertussis*	Ion	2.5	357.4	Fiona	3	32(8)	0.728	-
SRR443373	*C. elegans* CB4856	Illum.	83	16600	Fiona	22.27	445.06(8)	0.252	-
SRR492060	*D. melanogaster*	Illum.	25.9	3900	Fiona	33.55	108.12(8)	0.313	-
SRX016210	*D. melanogaster*	454	4.7	2600	Fiona	18	240.7(8)	0.646	-
SRR611140	*E. coli* K12 MG1655	Ion	4.66	754.1	Fiona	9	118.3(8)	0.812	-
SRR620425	*E. coli* K12 MG1655	Ion	4.23	721.8	Fiona	8	49.2(8)	0.74	-
SRR254209	*E. coli* O104	Ion	0.97	177.7	Fiona	2	15.2(8)	0.693	-
SRR1238539	*H. sapiens*	Ion	186.13	32900	Fiona	244	1187.1(8)	0.466	-
ERR161543	*P. falciparum* 3D7	Ion	1.95	307.7	Fiona	3	20.5(8)	0.541	-
ERR005143	*P. syringae* B728a	Illum.	3.55	255.7	Fiona	1.63	2.73(8)	0.913	-
ERR236069	*S. aureus*	Ion	1.33	355.9	Fiona	3	43.7(8)	0.649	-
SRR070596	*S. aureus* 21275	454	0.18	95.9	Fiona	1	12.3(8)	0.698	-
SRX039441	*S. cerevisae* UC5	454	0.69	192.1	Fiona	2	13.1(8)	0.36	-
NIDS4(o)	*D. melanogaster* (sim)	Illum.			Qamar(l)	0.2	0.8(-)	0.983	-

Table D.1 Continued from previous page

Dataset	Genome	Tech	# Reads Mil**	# Bases Mil**	Program	Mem GB	Rt mins (Num Th)	Gain	N50 (kb)*
NIDS5(o)	*B. vulgaris*	Illum.			Qamar(l)	1.5	19.06(-)	0.574	-
SRX001814	Acinetobacter ADP1	Illum.	18.1	652.7	Qamar(l)	8.6	23.08(-)	0.418	-
SRR001665_2	*E. coli* K12 MG1655	Illum.	10.4	749.4	Qamar(l)	4.5	58.8(-)	0.913	-
SRR088759	*L. lactis* NZ5522	Illum.	4.37	157.3	RACER(f)	0.4	2.61(1)	0.92	-
SRR361468	*T. pallidum*	Illum.	7.13	249.7	RACER(f)	0.4	7.7(1)	0.923	-
SRR396536	*E. coli* 75a	Illum.	3.45	259.1	RACER(f)	1.3	15.85(1)	0.839	-
DRR000852	*B. subtilis* 168	Illum.	1.75	264	RACER(f)	1.3	14.68(1)	0.935	-
SRR396532	*E. coli* 75b	Illum.	4.34	325.6	RACER(f)	1.4	19.78(1)	0.778	-
SRR396641	*P. aeruginosa* MPAO1	Illum.	9.3	335	RACER(f)	1	8.3(1)	0.895	-
SRR353563	*L. interrogans* L	Illum.	3.53	706.6	RACER(f)	0.8	27.75(1)	0.908	-
SRR397962	*L. interrogans* C	Illum.	3.56	712.7	RACER(f)	0.8	21.96(1)	0.892	-
SRR065202	*H. influenzae* KW20	Illum.	11.96	1000	RACER(f)	0.8	21.15(1)	0.843	-
NIDS6(o) / ERX002508	*E. coli* K12 MG1655	PacBio / Illum	/ 14.21	100 / 2800	proovread	19.6	1158(-)	-	-
NIDS7(o) / SRX158552	*A. thaliana*	PacBio / Illum	/ 43.21	128 / 9500	proovread	43.6	249120(-)	-	-
NIDS8(o) / SRX246904	*H. sapiens*	PacBio / Illum	/ 77.1	393 / 15600	proovread	40.1	3294720(-)	-	-
SRX246905			154.2	31200					
SRX246906			118.6	23900					
SRX247361			182.2	36800					
SRX247362(a)			78.6	15900					

Table D.1 Continued from previous page

Dataset	Genome	Tech	# Reads Mil**	# Bases Mil**	Program	Mem GB	Rt mins (Num Th)	Gain	N50 (kb)*
NIDS9(o) / ERX011200 ERX011186	*H. sapiens*	PacBio / Illum	/ 73.51 64.31	138 / 7400 4800	proovread	8.1	5040(-)	-	-
SRR000870	*E. coli* UTI89	454	0.21	55.6	Hector(e)	-	2.83(2)	0.853	-
SRR639330	*E. coli* O104	454	0.59	370.9	Hector(e)	-	8(2)	0.955	-
SRR957993	*S. enterica* ATCC 51955	454	0.18	101.2	Hector(e)	-	4(2)	0.882	-
NIDS10(o) / ERR022075	*E. coli* K12 MG1655	PacBio / Illum	0.03 / 2.3	98 / 231	LoRDEC	0.96	10(16)	0.899	-
NIDS11(o) / SRR567755	*S. cerevisiae* W303	PacBio / Illum	0.26 / 4.5	1500 / 450	LoRDEC	0.9	217(16)	0.819	-
ERR244164 ERR244165 ERR244166 / ERR244156	*M. undulatus*	PacBio / Illum	4.17 / 345.09	6800 / 35000	LoRDEC(i)	4.61	1747(48)	0.854	-
ERR330008	*P. aeruginosa* PAO1	Illum.	5.09	1400	Blue(b,e)	1.3+0.5	4.4+6.3(8)	-	132(V)
SRR352384	*S. cerevisiae* S288C	Illum.	26.03	4000	Trowel	10.4	14.1(32)	0.152	23.2(S) 36.8(V)
SRR060098	*D. melanogaster*	Illum.	18.96	3600	Trowel	19.3	18.4(32)	0.421	7.8(S) 8.9(V)
SRR018294	*D. melanogaster*	Illum.	9.46	1400	Trowel	15.4	8.4(32)	0.574	0.7(S) 0.5(V)
SRR018292	*D. melanogaster*	Illum.	12.24	1100	Trowel	5.4	4.2(32)	0.585	0.3(S) 0.2(V)
SRR018293	*D. melanogaster*	Illum.	8.54	769	Trowel	4.1	3.1(32)	0.542	0.2(S) 0.1(V)
SRR067388	*H. sapiens* hg19 mRNA	Illum.	9.99	749.9	Trowel	-	-	0.168	-

Table D.1 Continued from previous page

Dataset	Genome	Tech	# Reads Mil**	# Bases Mil**	Program	Mem GB	Rt mins (Num Th)	Gain	N50 (kb)*
SRR067389	*H. sapiens* hg19 mRNA	Illum.	17.77	1300	Trowel	-	-	0.114	-
SRR067390	*H. sapiens* hg19 mRNA	Illum.	11.57	868	Trowel	-	-	0.122	-
SRA000156	*E. coli* UTI89	454	0.45	276.5	HSHREC	3	12.23(-)	-	4.3(V)
NIDS1(o)	*E. coli* DH10B	SOLiD	2.83	141.5	HSHREC	3.7	41.11(-)	-	0.5(V)
SRR022918_1	*E. coli* 47	Illum.	7.2	677.2	CloudRS	-	-	0.22	17.7(V)
SRA048664	*E. coli* O104	Illum.	5.8	1800	Pollux	-	3.01(-)	-	37(V)
SRR023794 / SRR023796	*H. pylori* strain V225d	454	0.27	65	Karect	1.47	1.13(12)	0.956	-
SRR017972 / SRR029606	*Z. mobilis* subsp. mobilis ZM4	454	0.21	40	Karect	0.92	1.17(12)	0.946	-
B22-730	*E. coli* DH10B	Ion	0.49	160	Karect	3.61	4.23(12)	0.955	-
Gage S. Sapiens chr14	S. Sapiens	Illum.	36	3686	Karect	81.30	47.48(12)	0.832	-
SRR519926	*E. coli*	Illum.	0.8	201	ACE	6.2(q)	34.23(16)(q)	-(r)	-
SRR1200797	*M. tuberculosis*	Illum.	1.4	348.2	ACE	1.8(q)	18.21(16)(q)	-(r)	-
SRR1203044	*S. enterica*	Illum.	1.7	433.1	ACE	2.6(q)	27.13(16)(q)	-(r)	-
SRR1206093	*S. enterica*	Illum.	1.9	472.2	ACE	3.3(q)	30.83(16)(q)	-(r)	-
SRR1198952	*L. monocytogenes*	Illum.	2.1	507.7	ACE	2.7(q)	29.4(16)(q)	-(r)	-
SRR1119292	*P. syringae*	Illum.	2.5	639.8	ACE	3.6(q)	41.53(16)(q)	-(r)	-
SRR1151311	*B. dentium*	Illum.	3.9	984.2	ACE	2.6(q)	53.96(16)(q)	-(r)	-
SRR522163	*E. coli*	Illum.	11.1	2806.5	ACE	36(q)	387.76(16)(q)	-(r)	-
SRR1202083	*O. tsutsugamushi*	Illum.	10.3	3104.9	ACE	15.3(q)	291.86(16)(q)	-(r)	-
ERR400373	*M. tuberculosis*	Illum.	2	316	ACE	1.6(q)	22.54(16)(q)	-(r)	-
ERR230402	*S. enterica*	Illum.	3.2	325.7	ACE	1.3(q)	31.37(16)(q)	-(r)	-
ERR422544	*S. cerevisiae*	Illum.	4.7	477.6	ACE	2.4(q)	41.47(16)(q)	-(r)	-
SRR801797	*L. pneumophila*	Illum.	8.8	885	ACE	5(q)	86.73(16)(q)	-(r)	-

Table D.1 Continued from previous page

Dataset	Genome	Tech	# Reads Mil**	# Bases Mil**	Program	Mem GB	Rt mins (Num Th)	Gain	N50 (kb)*
SRR1191655	*E. coli*	Illum.	11.7	1184.3	ACE	2.9(q)	94.15(16)(q)	-(r)	-
SRR490124	*E. coli*	Illum.	21.5	2155.3	ACE	9.6(q)	130(16)(q)	-(r)	-
SRX218989	*C. elegans*	Illum.	31.6	3164.2	ACE	21.2(q)	262.1(16)(q)	-(r)	-
SRR543736	*C. elegans*	Illum.	57.7	5829.8	ACE	28.1(q)	443.03(16)(q)	-(r)	-
SRR823377	*D. melanogaster*	Illum.	63	6301.4	ACE	29.9(q)	450.55(16)(q)	-(r)	-
SRR988075	*D. melanogaster*	Illum.	75.9	7669.7	ACE	35.5(q)	533(16)(q)	-(r)	-
ERX069715	*H. sapiens*	Illum.	1357.7	137132.9	ACE	190.6(q)	8982.2(16)(q)	-(r)	-
ERX069504	*H. sapiens*	Illum.	1637.8	165419.5	ACE	224.9(q)	10807.4(16)(q)	-(r)	-
ERX069505	*H. sapiens*	Illum.	1708.1	172525.1	ACE	227.7(q)	11271.6(16)(q)	-(r)	-

** - information taken from NCBI SRA service when available, extracted by us or copied from the original article

* - The letters between parenthesis stand for: V - Velvet, S - SoapdeNovo, E - Edena, Sg - SGA, A - ABySS, C - Celera, A - ALLPATHS-LG

a - PacBio corrected with a sample at 50x coverage from all Illum. datasets

b - Resources used for preparing the dataset + resources used for correcting the data

c - **DecGPU** also needs a GPU, the table contains the result for 1xGPU execution

d - **CUDA-EC** also needs a GPU, the table contains the result for 1xGPU execution and 1xCPU

e - Approximate values for runtime extracted from the chart in the paper

f - The dataset was filtered using BWA

g - The dataset has a 30x, sampled at random from the original

h - Runtime for d (Hamming distance between two k-mers) being 1 and 2

i - **LoRDEC** ran on three servers, memory usage for one server, running time for all three in parallel with all CPUs used

j - Combined running time of the sequential counting on the CPU and parallel error correction on the GPU

Table D.1 Continued from previous page

Dataset	Genome	Tech	# Reads Mil**	# Bases Mil**	Program	Mem GB	Rt mins (Num Th)	Gain	N50 (kb)*

l - Gain calculated by us using the values for TP, FP, FN given by the authors and the formula in [1]
k - Profiles T1 - With all mapped reads; T2 - excluding reads ¿ 10 errors
m - Different profiles for the aligner and k-mer = 10
n - The authors do not explicitly pinpoint the dataset, it's our inference
o - NIDS - Non-Indexed DataSets - datasets that don't have a short name from an index engine like the NCBI SRA repo
p - Value of the corrected N50 as obtained by using the GAGE[78] approach
q - Approximate values calculated by us from the datasets info and the numbers per Mb given by the authors
r - Due to non-standard per base approach and multiple types of gain, we omit the results for ACE
s - N50 for scaffolds

The following datasets from Table D.1 were denoted with NIDS followed by an index:

- NIDS1 - http://solidsoftwaretools.com/gf/project/ecoli2x50/
- NIDS2 - http://genomic.ch/edena/mw2Reads.seq.gz
- NIDS3 - http://clcbio.com/index.php?id=1290
- NIDS4 - http://sharcgs.molgen.mpg.de/data/AC006575.reads.gz
- NIDS5 - http://sharcgs.molgen.mpg.de/data/ZR-47B15.reads_prb.gz http://sharcgs.molgen.mpg.de/data/ZR-47B15.reads_seq.gz
- NIDS6 - https://github.com/PacificBiosciences/DevNet/wiki/E%20coli%20K12%20MG1655%20Resequencing
- NIDS7 - PacBio_DevNet_Arabidopsis_P5C3_N50_8109_bp
- NIDS8 - https://github.com/PacificBiosciences/DevNet/wiki/H.-sapiens-10x-Sequence-Coverage-with-PacBio-data
- NIDS9 - http://www.stanford.edu/~kinfai/human_cerebellum_PacBioLR.zip
- NIDS10 - https://github.com/PacificBiosciences/DevNet/wiki/E%20coli%20K12%20MG1655%20Hybrid%20Assembly
- NIDS11 - https://github.com/PacificBiosciences/DevNet/wiki/Saccharomyces-cerevisiae-W303-Assembly-Contigs

- NDIS12 - http://sharcgs.molgen.mpg.de/data/reads_seq.gz
  http://sharcgs.molgen.mpg.de/data/reads_prb.gz

Table D.2: Testing configurations used by the authors

Program	CPU	RAM(GB)	OS	Notes
Blue	32xIntel E5-4640 cores @ 2.4GHz	512	Ubuntu	
DecGPU	2xIntel Xeon E5506 CPUs @ 2.13 GHz	16	Linux	
DecGPU	AMD Opteron 2378 CPUs @ 2.4 GHz	8	Linux	Node from a cluster; video card: NVIDIA Tesla S1070
SHREC	Intel Xeon @ 2.6 GHz	-	-	
CUDA-EC	AMD Opteron dual core @ 2.2 GHz	2	Fedora 8	video card: NVIDIA GeForce GTX 280
proovread	32 cores	192	-	HPC
proovread	4 cores	8	-	Single grid node; Correction of the genomic human dataset
Hector	2xIntel Xeon X5650 @ 2.66 GHz	96	Ubuntu 12.04	
Trowel	64xAMD Opteron 6274 cores @2.2 GHz	512	Linux	
Coral	4xAMD Opteron CPUs @2.6 GHz	32	Ubuntu 8.04	
Qamar	8xIntel Core i7 cores @2.93GHz	11.8	Ubuntu	
Musket	2xIntel Xeon X5650 CPUs @2.67 GHz	96	Ubuntu 12.04	
Par. Reptile	2xAMD Barcelona 3 (quad core) CPUs @ 2.3 GHz	8	Debian	Node in a cluster; SunFire X2200 blade; Authors used up to 128 nodes
RACER	24xAMD Opteron cores @ 2.1 GHz	98	Red Hat, CentoOS 5.5m	
LoRDEC	16 cores @ 2.53 GHz	32	Linux	Multiple servers with this configuration
CUDA-EC2	Intel i7 @2.67GHz	8	Fedora 10	Video card: NVIDIA Tesla C1060
BLESS	2xintel Xeon X5650 @2.67GHz	24	Scientific Linux	
Fiona	8xIntel Xeon X5550 @2.67Ghz	72	Debian 6	
Fiona	32 virtual cores	370		Configuration used for SRR1238539
Reptile	2xAMD Barcelona 3 (quad core) CPUs @ 2.3 GHz	8	Debian	SunFire X2200 blade
HSHREC	8xIntel Xeon cores @3.66GHz	32	-	JVM 1.6
REDEEM	Intel Xeon @ 3.16 GHz	-	-	-
Pollux	Intel Core i7-3820 @ 3.60 GHz	64	Linux	-
FADE	16xIntel Xeon X5650	24	-	-
Chung *et al.*	8xIntel Xeon @ 2.33 GHz cores	16	-	10 nodes in a Hadoop Cluster
Chung *et al.*	8xIntel Xeon @ 2.33 GHz cores	64	-	Single machine
ACE	16xIntel Xeon ES-2667 cores @ 2.9 GHz	256	SUSE 3.8.6-2	-

Bibliography

[1] Yang X, Chockalingam SP, Aluru S. A survey of error-correction methods for next-generation sequencing. Briefings in bioinformatics. 2013;14(1):56–66.

[2] Tahir M, Sardaraz M, Ikram AA, Bajwa H. Review of Genome Sequence Short Read Error Correction Algorithms. American Journal of Bioinformatics Research. 2013;3(1):1–9.

[3] Zimin A, Stevens KA, Crepeau MW, Holtz-Morris A, Koriabine M, Marçais G, et al. Sequencing and assembly of the 22-Gb loblolly pine genome. Genetics. 2014;196(3):875–890.

[4] Nystedt B, Street NR, Wetterbom A, Zuccolo A, Lin YC, Scofield DG, et al. The Norway spruce genome sequence and conifer genome evolution. Nature. 2013;497(7451):579–584.

[5] Friz CT. The biochemical composition of the free-living Amoebae¡ i¿ Chaos chaos, amoeba dubia¡/i¿ and¡ i¿ Amoeba proteus¡/i¿. Comparative biochemistry and physiology. 1968;26(1):81–90.

[6] Ilie L, Fazayeli F, Ilie S. HiTEC: accurate error correction in high-throughput sequencing data. Bioinformatics. 2011;27(3):295–302.

[7] Ross MG, Russ C, Costello M, Hollinger A, Lennon NJ, Hegarty R, et al. Characterizing and measuring bias in sequence data. Genome Biol. 2013;14(5):R51.

[8] Loman NJ, Misra RV, Dallman TJ, Constantinidou C, Gharbia SE, Wain J, et al. Performance comparison of benchtop high-throughput sequencing platforms. Nature biotechnology. 2012;30(5):434–439.

[9] Kelley DR, Schatz MC, Salzberg SL. Quake: quality-aware detection and correction of sequencing errors. Genome Biol. 2010;11(11):R116.

[10] Greenfield P, Duesing K, Papanicolaou A, Bauer DC. Blue: correcting sequencing errors using consensus and context. Bioinformatics. 2014;p. btu368.

[11] Kao WC, Chan AH, Song YS. ECHO: a reference-free short-read error correction algorithm. Genome research. 2011;21(7):1181–1192.

[12] Zhao Z, Yin J, Li Y, Xiong W, Zhan Y. An efficient hybrid approach to correcting errors in short reads. Modeling Decision for Artificial Intelligence. 2011;p. 198–210.

[13] Liu Y, Schröder J, Schmidt B. Musket: a multistage k-mer spectrum-based error corrector for Illumina sequence data. Bioinformatics. 2013;29(3):308–315.

[14] Liu Y, Schmidt B, Maskell DL. CUSHAW: a CUDA compatible short read aligner to large genomes based on the Burrows–Wheeler transform. Bioinformatics. 2012;28(14):1830–1837.

[15] Salmela L, Schröder J. Correcting errors in short reads by multiple alignments. Bioinformatics. 2011;27(11):1455–1461.

[16] Qu W, Hashimoto Si, Morishita S. Efficient frequency-based de novo short-read clustering for error trimming in next-generation sequencing. Genome research. 2009;19(7):1309–1315.

[17] marketsandmarkets com. Next Generation Sequencing (NGS) Market by Platforms (Illumina HiSeq, MiSeq, HiSeqX

Ten, NextSeq 500,Thermo Fisher Ion Proton/PGM), Bioinformatics (Exome Sequencing, RNA-Seq, ChIP-Seq), Technology (SBS, SMRT) & by Application (Diagnostics, Personalized Medicine) – Global Forecast to 2020; 2014. Accessed: 2014-09-09. .

[18] Illumina. Human whole-genome sequencing power; 2015. Available from: http://www.illumina.com/systems/hiseq-x-sequencing-system.html.

[19] Check Hayden E. Is the $1,000 genome for real?; 2015. Available from: http://www.nature.com/news/is-the-1-000-genome-for-real-1.14530.

[20] Institute NHGR. International Human Genome Sequencing Consortium Announces "Working Draft" of Human Genome; 2000. Available from: https://www.genome.gov/10001457.

[21] Sanger F, Nicklen S, Coulson AR. DNA sequencing with chain-terminating inhibitors. Proceedings of the National Academy of Sciences. 1977;74(12):5463–5467.

[22] Illumina. Specification Sheet: Illumina HiSeq X Series; 2015. Available from: http://www.illumina.com/documents/products/datasheets/datasheet-hiseq-x-ten.pdf.

[23] Illumina. Specification Sheet: HiSeq 3000/HiSeq 4000 Sequencing Systems; 2015. Available from: https://www.illumina.com/content/dam/illumina-marketing/documents/products/datasheets/hiseq-3000-4000-specification-sheet-770-2014-057.pdf.

[24] Illumina. Specification Sheet: NextSeq Series Specifications; 2015. Available from: http://www.illumina.

com/content/dam/illumina-marketing/documents/products/datasheets/datasheet-nextseq-500.pdf.

[25] Illumina. HiSeq 2500 Specifications; 2015. Available from: http://www.illumina.com/systems/hiseq_2500_1500/performance_specifications.html.

[26] Illumina. Specification Sheet: HiScan SQ System; 2012. Available from: http://www.illumina.com/documents/products/datasheets/datasheet_hiscansq.pdf.

[27] Illumina. Specification Sheet: Genome AnalyzerIIx System; 2011. Available from: https://support.illumina.com/content/dam/illumina-marketing/documents/products/datasheets/datasheet_genome_analyzeriix.pdf.

[28] Illumina. Specification Sheet: Genome AnalyzerIIe System; 2010. Available from: http://www.illumina.com/documents/products/datasheets/datasheet_genome_analyzer_IIe.pdf.

[29] Illumina. Specification Sheet: Genome AnalyzerII System; 2009. Available from: http://tucf.org/htseq_GenomeAnalyzer_SpecSheet.pdf.

[30] Illumina. Specification Sheet: Genome Analyzer System; 2007. Available from: http://www.geneworks.com.au/library/GenomeAnalyzer_SpecSheet.pdf.

[31] Illumina. Specification Sheet: MiSeq System; 2015. Available from: http://www.illumina.com/content/dam/illumina-marketing/documents/products/datasheets/datasheet_miseq.pdf.

[32] Scientific TF. Ion Proton System Specifications; 2015. Available from: http://www.thermofisher.com/es/en/home/life-science/sequencing/next-generation-sequencing/ion-torrent-next-generation-sequencing-workflow/ion-torrent-next-generation-sequencing-run-sequence/ion-proton-system-for-next-generation-sequencing/ion-proton-system-specifications.html.

[33] Scientific TF. Ion PGM System Specifications; 2015. Available from: http://www.thermofisher.com/es/en/home/life-science/sequencing/next-generation-sequencing/ion-torrent-next-generation-sequencing-workflow/ion-torrent-next-generation-sequencing-run-sequence/ion-pgm-system-for-next-generation-sequencing/ion-pgm-system-specifications.html.

[34] Scientific TF. 5500 W & 5500xl W Series; 2015. Available from: https://www.thermofisher.com/es/en/home/life-science/sequencing/next-generation-sequencing/solid-next-generation-sequencing/solid-next-generation-sequencing-systems-reagents-accessories.html.

[35] Scientific TF. 5500 & 5500xl Series; 2013. Available from: http://www.appliedbiosystems.com/absite/us/en/home/applications-technologies/solid-next-generation-sequencing/next-generation-systems.html.

[36] Scientific TF. SOLiD System accuracy with the Exact Call Chemistry module; 2011. Available from:

https://tools.thermofisher.com/content/sfs/brochures/cms_091372.pdf.

[37] Biosciences P. PacBio RS II; 2015. Available from: http://files.pacb.com/pdf/PacBio_RS_II_Brochure.pdf.

[38] Biosciences P. PacBio RS; 2015. Available from: http://files.pacb.com/pdf/PacBio_RS_Brochure.pdf.

[39] 454 R. GS FLX+ System; 2015. Available from: http://454.com/products/gs-flx-system/index.asp.

[40] 454 R. GS Junior System; 2015. Available from: http://454.com/products/gs-junior-system/index.asp.

[41] 454 R. GS Junior+ System; 2015. Available from: http://454.com/products/gs-junior-plus-system/index.asp.

[42] Technologies ON. Specification MinION & PromethION; 2015. Available from: https://www.nanoporetech.com/community/specifications.

[43] Fuller CW, Middendorf LR, Benner SA, Church GM, Harris T, Huang X, et al. The challenges of sequencing by synthesis. Nature biotechnology. 2009;27(11):1013–1023.

[44] Cock PJ, Fields CJ, Goto N, Heuer ML, Rice PM. The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. Nucleic acids research. 2010;38(6):1767–1771.

[45] Biosystems A. Overview of SOLiD^{TM} Sequencing Chemistry; 2013. Available from: http://www.appliedbiosystems.com/absite/us/en/home/applications-technologies/solid-next-generation-sequencing/

next-generation-systems/solid-sequencing-chemistry.html.

[46] Biosciences P. How does SMRT work?; 2014. Available from: http://www.pacificbiosciences.com/products/smrt-technology/.

[47] Zhu P, Craighead HG. Zero-mode waveguides for single-molecule analysis. Annual review of biophysics. 2012;41:269–293.

[48] Technologies ON. DNA: Nanopore sequencing; 2015. Available from: https://nanoporetech.com/applications/dna-nanopore-sequencing.

[49] Lieberman KR, Cherf GM, Doody MJ, Olasagasti F, Kolodji Y, Akeson M. Processive replication of single DNA molecules in a nanopore catalyzed by phi29 DNA polymerase. Journal of the American Chemical Society. 2010;132(50):17961–17972.

[50] Schulz MH, Weese D, Holtgrewe M, Dimitrova V, Niu S, Reinert K, et al. Fiona: a parallel and automatic strategy for read error correction. Bioinformatics. 2014;30(17):i356–i363.

[51] Mikheyev AS, Tin MM. A first look at the Oxford Nanopore MinION sequencer. Molecular ecology resources. 2014;14(6):1097–1102.

[52] Wirawan A, Harris RS, Liu Y, Schmidt B, Schröder J. HECTOR: a parallel multistage homopolymer spectrum based error corrector for 454 sequencing data. BMC bioinformatics. 2014;15(1):131.

[53] Wang XV, Blades N, Ding J, Sultana R, Parmigiani G. Estimation of sequencing error rates in short reads. BMC bioinformatics. 2012;13(1):185.

[54] Sahli M, Shibuya T. Qamar–A More Accurate DNA Sequencing Error Correcting Algorithm. International Proceedings of Chemical, Biological & Environmental Engineering. 2012;31.

[55] Bragg LM, Stone G, Butler MK, Hugenholtz P, Tyson GW. Shining a light on dark sequencing: characterising errors in Ion Torrent PGM data. PLoS computational biology. 2013;9(4):e1003031.

[56] Nakamura K, Oshima T, Morimoto T, Ikeda S, Yoshikawa H, Shiwa Y, et al. Sequence-specific error profile of Illumina sequencers. Nucleic acids research. 2011;p. gkr344.

[57] Schirmer M, Ijaz UZ, D'Amore R, Hall N, Sloan WT, Quince C. Insight into biases and sequencing errors for amplicon sequencing with the Illumina MiSeq platform. Nucleic acids research. 2015;p. gku1341.

[58] Illumina. TruSeq® DNA PCR-Free Sample Preparation Kit; 2013. Available from: http://www.appliedbiosystems.com/absite/us/en/home/applications-technologies/solid-next-generation-sequencing/next-generation-systems/solid-sequencing-chemistry.html.

[59] Biosciences S. Accel-NGS® DNA Library Kit for Ion Torrent; 2015. Available from: http://www.swiftbiosci.com/products/accel-ngs-dna-library-kit.

[60] Gen G. Fast, PCR free DNA library construction for 454; 2015. Available from: http://resource.jerei.com/10896/13041512044595_0.pdf.

[61] Sleep JA, Schreiber AW, Baumann U. Sequencing error correction without a reference genome. BMC bioinformatics. 2013;14(1):367.

[62] Wijaya E, Frith MC, Suzuki Y, Horton P. Recount: expectation maximization based error correction tool for next generation sequencing data. In: Genome Inform. vol. 23. World Scientific; 2009. p. 189–201.

[63] Dohm JC, Lottaz C, Borodina T, Himmelbauer H. Substantial biases in ultra-short read data sets from high-throughput DNA sequencing. Nucleic acids research. 2008;36(16):e105–e105.

[64] Schröder J, Schröder H, Puglisi SJ, Sinha R, Schmidt B. SHREC: a short-read error correction method. Bioinformatics. 2009;25(17):2157–2163.

[65] Shi H, Schmidt B, Liu W, Müller-Wittig W. A parallel algorithm for error correction in high-throughput short-read data on CUDA-enabled graphics hardware. Journal of Computational Biology. 2010;17(4):603–615.

[66] Shao W, Boltz VF, Spindler JE, Kearney MF, Maldarelli F, Mellors JW, et al. Analysis of 454 sequencing error rate, error sources, and artifact recombination for detection of Low-frequency drug resistance mutations in HIV-1 DNA. Retrovirology. 2013;10(1):18.

[67] Luo C, Tsementzi D, Kyrpides N, Read T, Konstantinidis KT. Direct comparisons of Illumina vs. Roche 454 sequencing technologies on the same microbial community DNA sample. PloS one. 2012;7(2):e30087.

[68] Gilles A, Meglécz E, Pech N, Ferreira S, Malausa T, Martin JF. Accuracy and quality assessment of 454 GS-FLX Titanium pyrosequencing. BMC genomics. 2011;12(1):245.

[69] Quail MA, Smith M, Coupland P, Otto TD, Harris SR, Connor TR, et al. A tale of three next generation sequencing platforms: comparison of Ion Torrent, Pacific Biosciences and Illumina MiSeq sequencers. BMC genomics. 2012;13(1):341.

[70] Koren S, Schatz MC, Walenz BP, Martin J, Howard JT, Ganapathy G, et al. Hybrid error correction and de novo assembly of single-molecule sequencing reads. Nature biotechnology. 2012;30(7):693–700.

[71] Salmela L, Rivals E. LoRDEC: accurate and efficient long read error correction. Bioinformatics. 2014;p. btu538.

[72] Hackl T, Hedrich R, Schultz J, Förster F. proovread: large-scale high accuracy PacBio correction through iterative short read consensus. Bioinformatics. 2014;p. btu392.

[73] Au KF, Underwood JG, Lee L, Wong WH. Improving PacBio long read accuracy by short read alignment. PLoS One. 2012;7(10):e46679.

[74] Technologies ON. The MinION™ device: a miniaturised sensing; 2014. Available from: https://nanoporetech.com/technology/the-minion-device-a-miniaturised-sensing-system/the-minion-device-a-miniaturised-sensing-system.

[75] Loman NJ, Quinlan AR. Poretools: a toolkit for analyzing nanopore sequence data. Bioinformatics. 2014;30(23):3399–3401.

[76] Watson M, Thomson M, Risse J, Talbot R, Santoyo-Lopez J, Gharbi K, et al. poRe: an R package for the visualization and analysis of nanopore sequencing data. Bioinformatics. 2014;p. btu590.

[77] Goodwin S, Gurtowski J, Ethe-Sayers S, Deshpande P, Schatz M, McCombie WR. Oxford Nanopore Sequencing and de novo Assembly of a Eukaryotic Genome. bioRxiv. 2015;p. 013490.

[78] Salzberg SL, Phillippy AM, Zimin A, Puiu D, Magoc T, Koren S, et al. GAGE: A critical evaluation of genome assemblies and assembly algorithms. Genome research. 2012;22(3):557–567.

[79] Molnar M, Ilie L. Correcting Illumina data. Briefings in Bioinformatics. 2014;p. bbu029.

[80] Pautasso M. Ten Simple Rules for Writing a Literature Review. PLoS computational biology. 2013;9(7):e1003149.

[81] Carnwell R, Daly W. Strategies for the construction of a critical review of the literature. Nurse education in practice. 2001;1(2):57–63.

[82] Keele S. Guidelines for performing systematic literature reviews in software engineering. Technical report, EBSE Technical Report EBSE-2007-01; 2007.

[83] Kitchenham B. Procedures for performing systematic reviews. Keele, UK, Keele University. 2004;33:2004.

[84] Aita T, Ichihashi N, Yomo T. Probabilistic model based error correction in a set of various mutant sequences analyzed by next-generation sequencing. Computational biology and chemistry. 2013;47:221–230.

[85] Prosperi MC, Salemi M. QuRe: software for viral quasispecies reconstruction from next-generation sequencing data. Bioinformatics. 2012;28(1):132–133.

[86] Macalalad AR, Zody MC, Charlebois P, Lennon NJ, Newman RM, Malboeuf CM, et al. Highly sensitive and specific detection of rare variants in mixed viral populations from massively parallel sequence data. PLoS computational biology. 2012;8(3):e1002417.

[87] Zagordi O, Bhattacharya A, Eriksson N, Beerenwinkel N. ShoRAH: estimating the genetic diversity of a mixed sample from next-generation sequencing data. BMC bioinformatics. 2011;12(1):119.

[88] Simpson JT, Durbin R. Efficient de novo assembly of large genomes using compressed data structures. Genome research. 2012;22(3):549–556.

[89] Zimin AV, Marçais G, Puiu D, Roberts M, Salzberg SL, Yorke JA. The MaSuRCA genome assembler. Bioinformatics. 2013;29(21):2669–2677.

[90] Luo R, Liu B, Xie Y, Li Z, Huang W, Yuan J, et al. SOAPdenovo2: an empirically improved memory-efficient short-read de novo assembler. Gigascience. 2012;1(1):18.

[91] Marçais G, Yorke JA, Zimin A. QuorUM: an error corrector for Illumina reads. arXiv preprint arXiv:13073515. 2013;.

[92] Golan D, Medvedev P. Using state machines to model the Ion Torrent sequencing process and to improve read error rates. Bioinformatics. 2013;29(13):i344–i351.

[93] Joppich M, Schmidl D, Bolger AM, Kuhlen T, Usadel B. PAGANtec: OpenMP Parallel Error Correction for Next-Generation Sequencing Data. In: OpenMP: Heterogenous Execution and Data Movements. Springer; 2015. p. 3–17.

[94] Genomeweb. Roche Shutting Down 454 Sequencing Business; 2013. Available from: http://www.genomeweb.com/sequencing/roche-shutting-down-454-sequencing-business.

[95] Pevzner PA, Tang H, Waterman MS. An Eulerian path approach to DNA fragment assembly. Proceedings of the National Academy of Sciences. 2001;98(17):9748–9753.

[96] Shi H, Schmidt B, Liu W, Muller-Wittig W. Accelerating error correction in high-throughput short-read DNA sequencing data with CUDA. In: Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on. IEEE; 2009. p. 1–8.

[97] Chaisson M, Pevzner P, Tang H. Fragment assembly with short reads. Bioinformatics. 2004;20(13):2067–2074.

[98] Yang X, Dorman KS, Aluru S. Reptile: representative tiling for short read error correction. Bioinformatics. 2010;26(20):2526–2533.

[99] Zhao X, Palmer LE, Bolanos R, Mircean C, Fasulo D, Wittenberg GM. EDAR: an efficient error detection and removal algorithm for next generation sequencing data. Journal of computational biology. 2010;17(11):1549–1560.

[100] Comaniciu D, Ramesh V, Meer P. The variable bandwidth mean shift and data-driven scale selection. In: Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on. vol. 1. IEEE; 2001. p. 438–445.

[101] Medvedev P, Scott E, Kakaradov B, Pevzner P. Error correction of high-throughput sequencing datasets with non-uniform coverage. Bioinformatics. 2011;27(13):i137–i141.

[102] Yang X, Aluru S, Dorman KS. Repeat-aware modeling and correction of short read errors. BMC bioinformatics. 2011;12(Suppl 1):S52.

[103] Liu Y, Schmidt B, Maskell DL. DecGPU: distributed error correction on massively parallel graphics processing units using CUDA and MPI. BMC bioinformatics. 2011;12(1):85.

[104] Shi H, Schmidt B, Liu W, Müller-Wittig W. Quality-score guided error correction for short-read sequencing data using CUDA. Procedia Computer Science. 2012;1(1):1129–1138.

[105] Shah AR, Chockalingam S, Aluru S. A parallel algorithm for spectrum-based short read error correction. In: Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International. IEEE; 2012. p. 60–70.

[106] Nikolenko SI, Korobeynikov AI, Alekseyev MA. BayesHammer: Bayesian clustering for error correction in single-cell sequencing. BMC genomics. 2013;14(Suppl 1):S7.

[107] Ilie L, Molnar M. RACER: Rapid and accurate correction of errors in reads. Bioinformatics. 2013;p. btt407.

[108] Song L, Florea L, Langmead B. Lighter: fast and memory-efficient error correction without counting. bioRxiv. 2014;.

[109] Heo Y, Wu XL, Chen D, Ma J, Hwu WM. BLESS: Bloom filter-based error correction solution for high-throughput sequencing reads. Bioinformatics. 2014;p. btu030.

[110] Milicchio F, Prosperi MC. HErCoOl: High-throughput Error Correction by Oligomers. In: Computer-Based Medical Systems (CBMS), 2014 IEEE 27th International Symposium on. IEEE; 2014. p. 227–232.

[111] Lim EC, Müller J, Hagmann J, Henz SR, Kim ST, Weigel D. Trowel: a fast and accurate error correction module for Illumina sequencing reads. Bioinformatics. 2014;p. btu513.

[112] Li H. BFC: correcting Illumina sequencing errors. Bioinformatics. 2015;p. btv290.

[113] Duma D, Cordero F, Beccuti M, Ciardo G, Close TJ, Lonardi S. Scrible: Ultra-Accurate Error-Correction of Pooled Sequenced Reads. In: Algorithms in Bioinformatics. Springer; 2015. p. 162–174.

[114] Sheikhizadeh S, de Ridder D. ACE: Accurate Correction of Errors using K-mer tries. Bioinformatics. 2015;p. btv332.

[115] Ramachandran A, Heo Y, Hwu Wm, Ma J, Chen D. FPGA accelerated DNA error correction. In: Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition. EDA Consortium; 2015. p. 1371–1376.

[116] Fan L, Cao P, Almeida J, Broder AZ. Summary cache: a scalable wide-area web cache sharing protocol. IEEE/ACM Transactions on Networking (TON). 2000;8(3):281–293.

[117] Marinier E, Brown DG, McConkey BJ. Pollux: platform independent error correction of single and mixed genomes. BMC bioinformatics. 2015;16(1):10.

[118] Gu Y, Liu X, Zhu Q, Dong Y, Brown CT, Pramanik S. A new method for DNA sequencing error verification and correction

via an on-disk index tree. In: Proceedings of the 6th ACM Conference on Bioinformatics, Computational Biology and Health Informatics. ACM; 2015. p. 503–504.

[119] Rani PJ, Srikanth G. An Efficient Indexing Method for Box Queries in NDDS Spaces using BoND-tree. IEEE TKDE. 2014;11(25):2629–2643.

[120] Miclotte G, Heydari M, Demeester P, Audenaert P, Fostier J. Jabba: Hybrid Error Correction for Long Sequencing Reads Using Maximal Exact Matches. In: Algorithms in Bioinformatics. Springer; 2015. p. 175–188.

[121] Salmela L. Correction of sequencing errors in a mixed set of reads. Bioinformatics. 2010;26(10):1284–1290.

[122] Zhao Z, Yin J, Zhan Y, Xiong W, Li Y, Liu F. PSAEC: an improved algorithm for short read error correction using partial suffix arrays. Frontiers in Algorithmics and Algorithmic Aspects in Information and Management. 2011;p. 220–232.

[123] Larsson NJ, Sadakane K. Faster suffix sorting. Theoretical Computer Science. 2007;387(3):258–272.

[124] Savel DM, LaFramboise T, Grama A, Koyutürk M. Suffix-Tree Based Error Correction of NGS Reads Using Multiple Manifestations of an Error. In: Proceedings of the International Conference on Bioinformatics, Computational Biology and Biomedical Informatics. ACM; 2013. p. 351.

[125] Chen CC, Chang YJ, Chung WC, Lee DT, Ho JM. CloudRS: An error correction algorithm of high-throughput sequencing data based on scalable framework. In: Big Data, 2013 IEEE International Conference on. IEEE; 2013. p. 717–722.

[126] Gnerre S, MacCallum I, Przybylski D, Ribeiro FJ, Burton JN, Walker BJ, et al. High-quality draft assemblies of mammalian genomes from massively parallel sequence data. Proceedings of the National Academy of Sciences. 2011;108(4):1513–1518.

[127] Chung WC, Chang YJ, Lee D, Ho JM. Using geometric structures to improve the error correction algorithm of high-throughput sequencing data on MapReduce framework. In: Big Data (Big Data), 2014 IEEE International Conference on. IEEE; 2014. p. 784–789.

[128] David M, Dzamba M, Lister D, Ilie L, Brudno M. SHRiMP2: sensitive yet practical short read mapping. Bioinformatics. 2011;27(7):1011–1012.

[129] Langmead B, Salzberg SL. Fast gapped-read alignment with Bowtie 2. Nature methods. 2012;9(4):357–359.

[130] Ono Y, Asai K, Hamada M. PBSIM: PacBio reads simulator—toward accurate genome assembly. Bioinformatics. 2013;29(1):119–121.

[131] Allam A, Kalnis P, Solovyev V. Karect: accurate correction of substitution, insertion and deletion errors for next-generation sequencing data. Bioinformatics. 2015;31(21):3421–3428.

[132] Schreiber AW, Shi BJ, Huang CY, Langridge P, Baumann U. Discovery of barley miRNAs through deep sequencing of short reads. BMC genomics. 2011;12(1):129.

[133] Do CB, Batzoglou S. What is the expectation maximization algorithm? Nature biotechnology. 2008;26(8):897–900.

[134] Beißbarth T, Hyde L, Smyth GK, Job C, Boon WM, Tan SS, et al. Statistical modeling of sequencing errors in SAGE libraries. Bioinformatics. 2004;20(suppl 1):i31–i39.

[135] Dempster AP, Laird NM, Rubin DB. Maximum likelihood from incomplete data via the EM algorithm. Journal of the Royal Statistical Society Series B (Methodological). 1977;p. 1–38.

[136] Yin X, Song Z, Dorman K, Ramamoorthy A. PREMIER—PRobabilistic error-correction using Markov inference in errored reads. In: Information Theory Proceedings (ISIT), 2013 IEEE International Symposium on. IEEE; 2013. p. 1626–1630.

[137] Yin X, Song Z, Dorman K, Ramamoorthy A. PREMIER Turbo: Probabilistic error-correction using Markov inference in errored reads using the turbo principle. In: Global Conference on Signal and Information Processing (GlobalSIP), 2013 IEEE. IEEE; 2013. p. 73–76.

[138] Artyomenko A, Mancuso N, Skums P, Mandoiu I, Zelikovsky A. kGEM: An Expectation Maximization Error Correction Algorithm for Next Generation Sequencing of Amplicon-based Data. In: Proc. International Symposium Bioinformatics Research and Applications; 2013. .

[139] Shen J, Cong Q, Grishin NV. The complete mitochondrial genome of Papilio glaucus and its phylogenetic implications. Meta gene. 2015;5:68–83.

[140] Wang C, Grohme MA, Mali B, Schill RO, Frohme M. Towards decrypting cryptobiosis-analyzing anhydrobiosis in the tardigrade milnesium tardigradum using transcriptome sequencing. PloS one. 2014;9(3).

[141] MacManes MD, Eisen MB. Characterization of the transcriptome, nucleotide sequence polymorphism, and natural selection in the desert adapted mouse Peromyscus eremicus. PeerJ. 2014;2:e642.

[142] Alkio M, Jonas U, Declercq M, Van Nocker S, Knoche M. Transcriptional dynamics of the developing sweet cherry (Prunus avium L.) fruit: sequencing, annotation and expression profiling of exocarp-associated genes. Horticulture Research. 2014;1.

[143] Gupta V, Markmann K, Pedersen CN, Stougaard J, Andersen SU. shortran: a pipeline for small RNA-seq data analysis. Bioinformatics. 2012;28(20):2698–2700.

[144] Henkel CV, Dirks RP, Jansen HJ, Forlenza M, Wiegertjes GF, Howe K, et al. Comparison of the exomes of common carp (Cyprinus carpio) and zebrafish (Danio rerio). Zebrafish. 2012;9(2):59–67.

[145] Dominova I, Sorokin D, Kublanov I, Patrushev M, Toshchakov S. Complete genome sequence of Salinarchaeum sp. strain HArcht-Bsk1T, isolated from hypersaline Lake Baskunchak, Russia. Genome announcements. 2013;1(4):e00505–13.

[146] Riedel T, Fiebig A, Petersen J, Gronow S, Kyrpides NC, Göker M, et al. Genome sequence of the Litoreibacter arenae type strain (DSM 19593T), a member of the Roseobacter clade isolated from sea sand. Standards in genomic sciences. 2013;9(1):117.

[147] Xu Q, Chen LL, Ruan X, Chen D, Zhu A, Chen C, et al. The draft genome of sweet orange (Citrus sinensis). Nature genetics. 2013;45(1):59–66.

[148] Jiao WB, Huang D, Xing F, Hu Y, Deng XX, Xu Q, et al. Genome-wide characterization and expression analysis of genetic variants in sweet orange. The Plant Journal. 2013;75(6):954–964.

[149] Lada AG, Stepchenkova EI, Waisertreiger I, Noskov VN, Dhar A, Eudy JD, et al. Genome-wide mutation avalanches induced in diploid yeast cells by a base analog or an APOBEC deaminase. PLoS Genet. 2013;9(9):e1003736.

[150] Fujimoto MS, Bodily PM, Okuda N, Clement MJ, Snell Q. Effects of error-correction of heterozygous next-generation sequencing data. BMC bioinformatics. 2014;15(Suppl 7):S3.

[151] Schatz MC, Maron LG, Stein JC, Wences AH, Gurtowski J, Biggers E, et al. Whole genome de novo assemblies of three divergent strains of rice, Oryza sativa, document novel gene space of aus and indica. Genome biology. 2014;15(11):506.

[152] Coates RC, Podell S, Korobeynikov A, Lapidus A, Pevzner P, Sherman DH, et al. Characterization of cyanobacterial hydrocarbon composition and distribution of biosynthetic pathways. PloS one. 2014;9(1).

[153] Rödelsperger C, Neher RA, Weller AM, Eberhardt G, Witte H, Mayer WE, et al. Characterization of genetic diversity in the nematode Pristionchus pacificus from population-scale resequencing data. Genetics. 2014;196(4):1153–1165.

[154] Nguyen TT, Hayes BJ, Ingram BA. Genetic parameters and response to selection in blue mussel (Mytilus galloprovincialis) using a SNP-based pedigree. Aquaculture. 2014;420:295–301.

[155] Nobu MK, Tamaki H, Kubota K, Liu WT. Metagenomic characterization of 'Candidatus Defluviicoccus tetraformis

strain TFO71', a tetrad-forming organism, predominant in an anaerobic–aerobic membrane bioreactor with deteriorated biological phosphorus removal. Environmental microbiology. 2014;16(9):2739–2751.

[156] Suzuki S, Horinouchi T, Furusawa C. Prediction of antibiotic resistance by gene expression profiles. Nature communications. 2014;5.

[157] Hill-Cawthorne GA, Hudson LO, El Ghany MFA, Piepenburg O, Nair M, Dodgson A, et al. Recombinations in Staphylococcal Cassette Chromosome ¡italic¿mec¡/italic¿ Elements Compromise the Molecular Detection of Methicillin Resistance in ¡italic¿Staphylococcus aureus¡/italic¿. PLoS ONE. 2014 06;9(6).

[158] Koch P, Platzer M, Downie BR. RepARK-de novo creation of repeat libraries from whole-genome NGS reads. Nucleic acids research. 2014;p. gku210.

[159] Busk PK, Lange M, Pilgaard B, Lange L. Several genes encoding enzymes with the same activity are necessary for aerobic fungal degradation of cellulose in nature. PloS one. 2014;9(12):e114138.

[160] Gilchrist AS, Shearman DC, Frommer M, Raphael KA, Deshpande NP, Wilkins MR, et al. The draft genome of the pest tephritid fruit fly Bactrocera tryoni: resources for the genomic analysis of hybridising species. BMC genomics. 2014;15(1):1153.

[161] Burke GR, Walden KKO, Whitfield JB, Robertson HM, Strand MR. Widespread Genome Reorganization of an Obligate Virus Mutualist. PLoS Genet. 2014 09;10(9).

[162] Kenny NJ, Namigai EK, Marlétaz F, Hui JH, Shimeld SM. Draft genome assemblies and predicted microRNA complements of the intertidal lophotrochozoans Patella vulgata (Mollusca, Patellogastropoda) and Spirobranchus (Pomatoceros) lamarcki (Annelida, Serpulida). Marine genomics. 2015;.

[163] Fitak RR, Mohandesan E, Corander J, Burger PA. The de novo genome assembly and annotation of a female domestic dromedary of North African origin. Molecular ecology resources. 2015;.

[164] Le Duc D, Renaud G, Krishnan A, Almén MS, Huynen L, Prohaska SJ, et al. Kiwi genome provides insights into evolution of a nocturnal lifestyle. Genome biology. 2015;16(1):1–15.

[165] Fiebig A, Loof TG, Babbar A, Itzek A, Koehorst JJ, Schaap PJ, et al. Comparative Genomics of Streptococcus pyogenes M1 isolates differing in virulence and propensity to cause systemic infection in mice. International Journal of Medical Microbiology. 2015;305(6):532–543.

[166] Lambert D, Carrillo CD, Koziol AG, Manninger P, Blais BW. GeneSippr: A Rapid Whole-Genome Approach for the Identification and Characterization of Foodborne Pathogens such as Priority Shiga Toxigenic ¡italic¿Escherichia coli¡/italic¿. PLoS ONE. 2015 04;10(4).

[167] Cong Q, Borek D, Otwinowski Z, Grishin NV. Tiger Swallowtail Genome Reveals Mechanisms for Speciation and Caterpillar Chemical Defense. Cell reports. 2015;10(6):910–919.

[168] Jünemann S, Prior K, Albersmeier A, Albaum S, Kalinowski J, Goesmann A, et al. GABenchToB: A Genome Assem-

bly Benchmark Tuned on Bacteria and Benchtop Sequencers. PLoS ONE. 2014 09;9(9).

[169] Walter MC, Öhrman C, Myrtennäs K, Sjödin A, Byström M, Larsson P, et al. Genome sequence of Coxiella burnetii strain Namibia. Standards in genomic sciences. 2014;9:22.

[170] Nikolaichik Y, Gorshkov V, Gogolev Y, Valentovich L, Evtushenkov A. Genome sequence of Pectobacterium atrosepticum strain 21A. Genome announcements. 2014;2(5).

[171] Engel P, Stepanauskas R, Moran NA. Hidden Diversity in Honey Bee Gut Symbionts Detected by Single-Cell Genomics. PLoS Genet. 2014 09;10(9).

[172] Aeschlimann SH, Jönsson F, Postberg J, Stover NA, Petera RL, Lipps HJ, et al. The draft assembly of the radically organized Stylonychia lemnae macronuclear genome. Genome biology and evolution. 2014;6(7):1707–1723.

[173] Kleigrewe K, Almaliti J, Tian IY, Kinnel RB, Korobeynikov A, Monroe EA, et al. Combining Mass Spectrometric Metabolic Profiling with Genomic Analysis: A Powerful Approach for Discovering Natural Products from Cyanobacteria. Journal of natural products. 2015;78(7):1671–1682.

[174] Grob C, Taubert M, Howat AM, Burns OJ, Dixon JL, Richnow HH, et al. Combining metagenomics with metaproteomics and stable isotope probing reveals metabolic pathways used by a naturally occurring marine methylotroph. Environmental microbiology. 2015;.

[175] Neumann AM, Balmonte JP, Berger M, Giebel HA, Arnosti C, Voget S, et al. Different utilization of alginate and other algal polysaccharides by marine Alteromonas macleodii ecotypes. Environmental microbiology. 2015;.

[176] Lada AG, Kliver SF, Dhar A, Polev DE, Masharsky AE, Rogozin IB, et al. Disruption of Transcriptional Coactivator Sub1 Leads to Genome-Wide Re-distribution of Clustered Mutations Induced by APOBEC in Active Yeast Genes. PLoS Genet. 2015 05;11(5).

[177] Boudreau PD, Monroe EA, Mehrotra S, Desfor S, Korobeynikov A, Sherman DH, et al. Expanding the Described Metabolome of the Marine Cyanobacterium Moorea producens JHB through Orthogonal Natural Products Workflows. PloS one. 2015;10(7).

[178] De Wit P, Pespeni MH, Palumbi SR. SNP genotyping and population genomics from expressed sequences–current advances and future possibilities. Molecular ecology. 2015;24(10):2310–2323.

[179] Taniguti LM, Schaker PD, Benevenuto J, Peters LP, Carvalho G, Palhares A, et al. Complete genome sequence of Sporisorium scitamineum and biotrophic interaction transcriptome with sugarcane. PloS one. 2015;10(6):e0129318.

[180] Li X, Fan D, Zhang W, Liu G, Zhang L, Zhao L, et al. Outbred genome sequencing and CRISPR/Cas9 gene editing in butterflies. Nature communications. 2015;6.

[181] D'Agostino N, Golas T, Van de Geest H, Bombarely A, Dawood T, Zethof J, et al. Genomic analysis of the native European Solanum species, S. dulcamara. BMC genomics. 2013;14(1):356.

[182] Ištvánek J, Jaroš M, Křenek A, Řepková J. Genome assembly and annotation for red clover (Trifolium pratense; Fabaceae). American journal of botany. 2014;101(2):327–337.

[183] Yang R, Dai Z, Chen S, Chen L. MicroRNA-mediated gene regulation plays a minor role in the transcriptomic plasticity of cold-acclimated zebrafish brain tissue. BMC genomics. 2011;12(1):605.

[184] Zawada AM, Rogacev KS, Rotter B, Winter P, Marell RR, Fliser D, et al. SuperSAGE evidence for CD14++ CD16+ monocytes as a third monocyte subset. Blood. 2011;118(12):e50–e61.

[185] Roccaro M, Ahmadinejad N, Colby T, Somssich IE. Identification of functional cis-regulatory elements by sequential enrichment from a randomized synthetic DNA library. BMC plant biology. 2013;13(1):164.

[186] Ollier M, Radosevic-Robin N, Kwiatkowski F, Ponelle F, Viala S, Privat M, et al. DNA repair genes implicated in triple negative familial non-BRCA1/2 breast cancer predisposition. American journal of cancer research. 2015;5(7):2113.

[187] Fertin G, Jean G, Radulescu A, Rusu I. Hybrid de novo tandem repeat detection using short and long reads. BMC medical genomics. 2015;8(Suppl 3):S5.

[188] Yan L, Wang X, Liu H, Tian Y, Lian J, Yang R, et al. The Genome of Dendrobium officinale Illuminates the Biology of the Important Traditional Chinese Orchid Herb. Molecular plant. 2014;.

[189] Weirather JL, Afshar PT, Clark TA, Tseng E, Powers LS, Underwood J, et al. Characterization of fusion genes and the significantly expressed fusion isoforms in breast cancer by hybrid sequencing. Nucleic Acids Research. 2015;p. 1.

[190] Mulley JF, Hargreaves AD. Snake venom gland cDNA sequencing using the Oxford Nanopore MinION portable DNA sequencer. bioRxiv. 2015;p. 025148.

[191] Engström PG, Tommei D, Stricker SH, Ender C, Pollard SM, Bertone P. Digital transcriptome profiling of normal and glioblastoma-derived neural stem cells identifies genes associated with patient survival. Genome medicine. 2012;4(10):1–20.

[192] Cox MP, Peterson DA, Biggs PJ. SolexaQA: At-a-glance quality assessment of Illumina second-generation sequencing data. BMC bioinformatics. 2010;11(1):485.

[193] Rizk G, Lavenier D, Chikhi R. DSK: k-mer counting with very low memory usage. Bioinformatics. 2013;p. btt020.

[194] Deorowicz S, Debudaj-Grabysz A, Grabowski S. Disk-based k-mer counting on a PC. BMC bioinformatics. 2013;14(1):160.

[195] Marçais G, Kingsford C. A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. Bioinformatics. 2011;27(6):764–770.

[196] Zerbino DR, Birney E. Velvet: algorithms for de novo short read assembly using de Bruijn graphs. Genome research. 2008;18(5):821–829.

[197] Bloom BH. Space/time trade-offs in hash coding with allowable errors. Communications of the ACM. 1970;13(7):422–426.

[198] Fichot EB, Norman RS. Microbial phylogenetic profiling with the Pacific Biosciences sequencing platform. Microbiome. 2013;1(1):10.

[199] Bentley DR, Balasubramanian S, Swerdlow HP, Smith GP, Milton J, Brown CG, et al. Accurate whole human genome sequencing using reversible terminator chemistry. Nature. 2008;456(7218):53–59.

[200] Hillier LW, Marth GT, Quinlan AR, Dooling D, Fewell G, Barnett D, et al. Whole-genome sequencing and variant discovery in C. elegans. Nature methods. 2008;5(2):183–188.

[201] Kozarewa I, Ning Z, Quail MA, Sanders MJ, Berriman M, Turner DJ. Amplification-free Illumina sequencing-library preparation facilitates improved mapping and assembly of (G+ C)-biased genomes. Nature methods. 2009;6(4):291–295.

[202] Yoo AB, Jette MA, Grondona M. SLURM: Simple linux utility for resource management. In: Job Scheduling Strategies for Parallel Processing. Springer; 2003. p. 44–60.

[203] Li R, Li Y, Kristiansen K, Wang J. SOAP: short oligonucleotide alignment program. Bioinformatics. 2008;24(5):713–714.

[204] Li H, Durbin R. Fast and accurate short read alignment with Burrows–Wheeler transform. Bioinformatics. 2009;25(14):1754–1760.

[205] Miller JR, Koren S, Sutton G. Assembly algorithms for next-generation sequencing data. Genomics. 2010;95(6):315–327.

[206] Bradnam KR, Fass JN, Alexandrov A, Baranay P, Bechner M, Birol I, et al. Assemblathon 2: evaluating de novo methods of genome assembly in three vertebrate species. GigaScience. 2013;2(1):1–31.

[207] Darling AE, Tritt A, Eisen JA, Facciotti MT. Mauve assembly metrics. Bioinformatics. 2011;27(19):2756–2757.

[208] Hernandez D, François P, Farinelli L, Østerås M, Schrenzel J. De novo bacterial genome sequencing: millions of very short reads assembled on a desktop computer. Genome research. 2008;18(5):802–809.

[209] Massingham T, Goldman N. simNGS and simLibrary–software for simulating next-gen sequencing data; 2012.

[210] Holtgrewe M. Mason–a read simulator for second generation sequencing data. Technical Report FU Berlin. 2010;.

[211] Huang W, Li L, Myers JR, Marth GT. ART: a next-generation sequencing read simulator. Bioinformatics. 2012;28(4):593–594.

[212] Dohm JC, Lottaz C, Borodina T, Himmelbauer H. SHARCGS, a fast and highly accurate short-read assembly algorithm for de novo genomic sequencing. Genome research. 2007;17(11):1697–1706.

[213] Patel RK, Jain M. NGS QC Toolkit: a toolkit for quality control of next generation sequencing data. PloS one. 2012;7(2):e30619.

[214] Schmieder R, Edwards R. Quality control and preprocessing of metagenomic datasets. Bioinformatics. 2011;27(6):863–864.

[215] Mozilla Foundation. 'Epic Citadel' Demo Shows the Power of the Web as a Platform for Gaming; 2013. Accessed: 2016-01-30. https://blog.mozilla.org/futurereleases/2013/05/02/epic-citadel-demo-shows-the-power-of-the-web-as-a-platform-

[216] Tilkov S, Vinoski S. Node. js: Using JavaScript to build high-performance network programs. IEEE Internet Computing. 2010;(6):80–83.

[217] Goecks J, Nekrutenko A, Taylor J, et al. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. Genome Biol. 2010;11(8):R86.

[218] Sims D, Sudbery I, Ilott NE, Heger A, Ponting CP. Sequencing depth and coverage: key considerations in genomic analyses. Nature Reviews Genetics. 2014;15(2):121–132.

[219] Peng Y, Leung HC, Yiu SM, Chin FY. IDBA–a practical iterative de Bruijn graph de novo assembler. In: Research in Computational Molecular Biology. Springer; 2010. p. 426–440.

[220] Simpson JT, Wong K, Jackman SD, Schein JE, Jones SJ, Birol I. ABySS: a parallel assembler for short read sequence data. Genome research. 2009;19(6):1117–1123.

[221] Hoffmann S, Otto C, Kurtz S, Sharma CM, Khaitovich P, Vogel J, et al. Fast mapping of short sequences with mismatches, insertions and deletions using index structures. PLoS computational biology. 2009;5(9):e1000502.

[222] KA W. DNA Sequencing Costs: Data from the NHGRI Genome Sequencing Program (GSP); 2014. Accessed: 2014-04-20. .

[223] Gotoh O. An improved algorithm for matching biological sequences. Journal of molecular biology. 1982;162(3):705–708.

[224] Aarno D, Zeek E, Smoot M. Templatized C++ Command Line Parser Library; 2013. Accessed: 2014-01-10. .

[225] Chaisson MJ, Tesler G. Mapping single molecule sequencing reads using basic local alignment with successive refinement (BLASR): application and theory. BMC bioinformatics. 2012;13(1):238.

[226] NCBI. The NCBI C++ Toolkit; 2014. Accessed: 2014-02-09. .

[227] YourKit. YourKit Java Profiler; 2014. Accessed: 2014-04-09. http://www.yourkit.com/overview/index.jsp.

[228] EJ-Technologies. EJ-Technologies JProfiler; 2014. Accessed: 2014-04-09. http://www.ej-technologies.com/download/jprofiler/files.

[229] Margulies M, Egholm M, Altman WE, Attiya S, Bader JS, Bemben LA, et al. Genome sequencing in microfabricated high-density picolitre reactors. Nature. 2005;437(7057):376–380.

[230] Kumar S, Blaxter ML. Comparing de novo assemblers for 454 transcriptome data. BMC genomics. 2010;11(1):571.

[231] Earl D, Bradnam K, John JS, Darling A, Lin D, Fass J, et al. Assemblathon 1: a competitive assessment of de novo short read assembly methods. Genome research. 2011;21(12):2224–2241.

[232] Robinson JT, Thorvaldsdóttir H, Winckler W, Guttman M, Lander ES, Getz G, et al. Integrative genomics viewer. Nature biotechnology. 2011;29(1):24–26.

[233] Rutherford K, Parkhill J, Crook J, Horsnell T, Rice P, Rajandream MA, et al. Artemis: sequence visualization and annotation. Bioinformatics. 2000;16(10):944–945.

[234] Chevreux B, Wetter T, Suhai S. Genome sequence assembly using trace signals and additional sequence information. In: German Conference on Bioinformatics; 1999. p. 45–56.