Algorithmes bio-informatiques pour l'analyse de données de séquençage à haut débit

New algorithmic and bioinformatic approaches for the analysis of data from high throughput sequencing

THÈSE

présentée et soutenue publiquement le 11 décembre 2013

pour l'obtention du

Doctorat de l'Université de Lille 1 – Sciences et Technologies

(spécialité informatique)

par

Evguenia Kopylova

Composition du jury

Rapporteurs :	Veli Mäkinen, Professor	University of Helsinki
	Thierry Lecroq, Professor	LITIS, University of Rouen
Examinateurs :	Laurent Noé, Assistant Professor, *co-encadrant de thèse*	LIFL, CNRS, Univ. Lille 1, Inria
	Olivier Jaillon, Researcher CEA	LABIS, Genoscope
	Joachim Niehren, DR Inria Lille	Inria
	Hélène Touzet, DR CNRS, *directrice de thèse*	LIFL, CNRS, Univ. Lille 1, Inria

UNIVERSITÉ DE LILLE 1 – SCIENCES ET TECHNOLOGIES
ÉCOLE DOCTORALE SCIENCES POUR L'INGÉNIEUR
Laboratoire d'Informatique Fondamentale de Lille — UMR 8022
U.F.R. d'I.E.E.A. – Bât. M3 – 59655 VILLENEUVE D'ASCQ CEDEX
Tél. : +33 (0)3 28 77 85 41 – Télécopie : +33 (0)3 28 77 85 37 – email : `direction@lifl.fr`

Contents

Acknowledgments

These three years of my doctoral studies frame one of the most challenging and exhilarating periods in my life. There hasn't been a day where my mind wasn't wrapped around an exciting new problem and the perpetual drive to conquer it. All of the work resulting from this thesis could not have been possible without the great support I received from my family, supervisors, friends, colleagues and my dearest fiancé. Here I would like to take the opportunity to thank these people.

Tout d'abord, je voudrais dire un grand merci à mes deux directeurs de thèse - Hélène Touzet et Laurent Noé. Nos fréquentes discussions sur les algorithmes et les codes développés dans cette thèse nous ont permis d'améliorer et d'accélérer nos recherches dans leur ensemble. Mon séjour en France était très agréable grâce à leur aide avec la langue, la culture et l'encouragement quotidien. En plus, je tiens à remercier tous les membres de l'équipe Bonsai ! D'une manière ou d'une autre chacun d'eux a eu une influence positive sur ma vie et je suis très reconnaissante pour cela. Un grand merci à Jean-Frédéric pour tous ces rires et les desserts savoureux qu'il a preparés !

Je tiens à remercier Corinne Da Silva, Eric Pelletier, Jean-Marc Aury et Olivier Jaillon de l'équipe LABIS au Genoscope pour les discussions fructueuses sur SortMeRNA et leurs suggestions utiles pour son amélioration.

I would like to thank Veli Mäkinen and Thierry Lecroq for their helpful suggestions and corrections to my manuscript, as well as Olivier Jaillon and Joachim Niehren for agreeing to participate in the thesis committee.

I would like to thank all of my friends here in Lille for making my stay so fun! Gracias Karen for being there from the start to the finish! Danke Anke and Philipp for introducing us to the Feuerzangenbowle and for hosting the international dinners! Dziękuję to Ewa for all the support and smiles you always radiate! 谢谢 to Di and Wei for always surprising us with new Chinese delicacies!

Я хочу поблагодарить всю мою семью за их любовь и постоянную поддержку. Несмотря на то, что все мои родные люди были далеко от меня, они всегда держали со мной тесный контакт через скаып и почтовые открытки. Без них мне было бы гораздо тяжелеые пройти тот путь, где я сейчас.

Finalmente, vorrei ringraziare il mio caro fidanzato, Fabio. La sua presenza e il sostegno durante questo dottorato ha liberato la mia mente mettendo da parte gli algoritmi di tanto in tanto per godersi la vita. Quest'uomo è sempre pieno di energia e non ho parole per esprimere il positivo effetto che ha avuto sulla mia vita. E così, la vita è bella!

Papers

Published

1. E. Kopylova, L. Noé and H. Touzet. SortMeRNA: fast and accurate filtering of ribosomal RNAs in metatranscriptomic data. Bioinformatics, 28(24):3211-7, 2012.

Submitted

2. E. Kopylova, L. Noé, C. Da Silva, J.-F. Berthelot, A. Alberti, J.-M. Aury and H. Touzet. Deciphering metatranscriptomic data. Editor E. Picardi. Book chapter in Methods in Molecular Biology. In Press, 2013.

3. E. Kopylova, L. Noé, M. Salson and H. Touzet. Universal read mapping of genomic and metagenomic data using SortMeDNA. (Submitted).

Introduction

A genome is a sequence of DNA nucleotides which encodes complex instructions to build and develop an organism, with a copy existing in every cell of the organism's body. Although an organism will commence with a genome inherited from its parents, throughout its lifetime, the genome will frequently experience changes, known as *mutations*, in the nucleotide sequence during replication and in response to the surrounding environment. These changes can manifest as single nucleotide polymorphisms (SNP) or through insertions or deletions of nucleotides in the genome. *Somatic* mutations occur in the non-reproductive cells of an organism and are not transmitted to the offspring. *Germ-line* mutations, on the other hand, occur in the gamete cells of an organism and are hereditary. If a mutation occurs in a protein coding region of the genome, it may alter the behavior of the protein and ordinarily cause harm to the organism. For example, a deletion of three nucleotides, namely the amino acid phenylalanine, in the human gene CFTR causes irregular folding of the synthesized protein and leads to a critical genetic disorder known as cystic fibrosis. Being able to discover, understand and cure such detrimental changes is one of the leading reasons for which biological sequence analysis is indispensable to humanity. On the contrary, sometimes mutations can lead to favorable changes, as for example the single nucleotide mutation which disables the production of the toxic hydrogen cyanide found in wild almonds, yielding to the sweet domesticated almonds popularly consumed today [Heppner, 1923].

Even more, the development of revolutionary sequencing technologies used to unveil the sequences of DNA molecules can now be applied to entire microbial communities in an exciting new field named *metagenomics*. One influential application of metagenomics is the diagnosis of pathogenic diseases, which can be used to detect candidate pathogens responsible for an infection, such as was shown in the discovery of bacteria *C. jejuni* causing acute diarrhea in an infected patient [Nakamura et al., 2008]. Another study in [Chan et al., 2013] focused on the stress responses of microbial communities in the McMurdo Dry Valleys of Antarctica in order to identify functional traits that motivate prosperity of these communities in the much threatened biome. In reality, applications of biological sequence analysis are innumerable, and new software and technologies are being developed every day to facilitate their studies.

The layout of nucleotides in a DNA molecule can be determined using sequencing technologies which exploit advanced techniques to read and record each nucleotide one letter at a time. The principal limitation to all of these methods is that the sequence cannot be read in one go, but must be randomly split into millions of overlapping fragments which are individually sequenced to produce *reads*. In the final step, the sequenced reads are either reassembled back into the original sequence, an approach known as *de novo* assembly, or aligned against an existing reference sequence, known as *read mapping*, using appropriate bioinformatic tools. With the arrival of next-generation sequencing technologies, gigabytes of DNA material can be sequenced in parallel within very short time periods [Qin et al., 2009]. Big data brings along big challenges for the management and analysis of millions (to even billions) of short sequenced reads [Wandelt et al., 2012]. Morever, different read lengths and sequencing error rates must be taken into account when developing bioinformatic tools applicable to data from multiple sequencing platforms. The objective of this thesis is to design and implement new algorithms for the analysis of data produced by (next-generation) high-throughput sequencing technologies (HTS). More precisely, we concentrate on algorithms for performing genomic sequence alignment, a common application in whole-genome resequencing, and also for aligning metatranscriptomic and metagenomic data, that is the RNA and DNA belonging to thousands of different species directly extracted from their environment.

Metatranscriptomics is the study of total RNA extracted from microorganisms in a community. Total RNA is a composition of many subgroups including messenger RNA (mRNA), ribosomal RNA (rRNA), transfer RNA (tRNA) and other small non-coding RNA. Given that the former three RNA groups (rRNA, mRNA and tRNA) are chief participants in the synthesis of proteins, studying them can help to determine the activity and composition of the constituting species. The concentration and diversity of mRNA present in a sample can shed light on the actively expressed genes at the period of sampling. Changes in the environment, such as nutrient availability, temperature fluctuations and stress, can heavily influence normal gene expression. By tracking these changes, scientists can identify the cause and effect of threatened or thriving ecosystems and take measures to restore them. Another group of closely studied RNA is rRNA, which makes up to 90% of total RNA and is responsible for translating mRNA into protein. Ribosomal RNA is an ideal candidate for taxonomic analyses due to its presence in all cells and highly conserved structure. Depending on the question asked, it is often required that the sequenced reads from total RNA are sorted into their appropriate subgroups prior to further analysis. In this thesis, we introduce a new software tool called SortMeRNA to rapidly filter rRNA reads from metatranscriptomic data using a reference database of known rRNAs.

Another leading application of NGS is metagenomics, the study of total DNA retrieved from an environment. Whereas metatranscriptomic data renders information on the actively transcribed genes in a community, metagenomics provides the entire collection of genes and all other noncoding DNA within. The main goal of metagenomics is to obtain the complete genome sequences of all organisms in order to study their individual roles, whole-genome species diversity and symbiotic relationships of a community. Nowadays, metagenomic studies are popular in medicine, agriculture, biotechnology and environmental genomics. Global projects such as Tara Oceans or Tara Oceans Polar Circle endeavor on yearly expeditions to study marine ecosystems by collecting plankton from waters all over the world. An important subgroup of marine plankton, phytoplankton, is responsible for producing half of the total oxygen on earth, consequently changes to its population will directly affect all other organisms higher in the food chain hierarchy. In [Boyce et al., 2010], it has been reported that nearly 40% of surface phytoplankton in the northern hemisphere have vanished since 1950, and our understanding of how to replenish this loss is vital to sustain a healthy planet. MetaHIT [Ehrlich, 2011] is another metagenomic project focusing on the human intestinal microbiota and its connection to health and disease. The MetaHIT consortium hosts a broad database of publicly available draft genomes and partial sequences for the bacteria commonly found in the human intestinal tract. One of the greatest challenges in such large-scale projects is the ability to store, sort through and analyze the enourmous amounts of sequenced data. In this thesis, we introduce a second software tool called SortMeDNA, which has been designed to align large-scale sequenced data against a database of known reference sequences with accuracy and speed.

Life on planet earth evolves at an accelerated pace as a result of our strong curiosity and innate vivacity for growth and expantion. Understanding the biological building blocks of organisms is not only useful for modern day activites, but can be in the future applied to stimulating life on other planets. The Space Biosciences Division at NASA is actively exploring areas of research such as atmosphere revitalization, sources for biofuel [Buckwalter et al., 2013] and biological systems capable of self-producing natural resources. Given the hostile environments on other planets such as Mars, any signs of life would probably resemble single-celled microorganisms such as archaea known today [Morozova et al., 2007], therefore their current study through metatranscriptomics and metagenomics may one day offer advantages beyond our own world.

⋆ ⋆ ⋆

Contribution

The works of this thesis were supported by the ANR Project MAPPI (grant ANR-2010-COSI-004). Project MAPPI is a three-year collaboration between the teams LIAFA at the University of Paris-Diderot, IRISA in Rennes and the French National Sequencing Center (Genoscope) in Évry to develop bioinformatic tools for the analysis of metatranscriptomic and metagenomic data produced by the Tara Oceans expedition.

In this thesis we develop a new type of *approximate* seed allowing up to 1 error: The error can either be a mismatch or an indel, and its position in the seed is not predetermined. This unique feature gives the seed flexibility for different error types, such as indels in 454 reads, unpredictable error distribution, as readily observed with PacBio reads and capturing similarities between distant related species. Furthermore, we introduce an indexing data structure specifically tailored to perform fast queries in large texts using this approximate seed. We show the efficiency of our method with two developed software tools, SortMeRNA for filtering ribosomal RNA from metatranscriptomic data (see Chapter 3), and SortMeDNA for mapping reads from metagenomic and genomic sequences generated by second- and third-generation technologies (see Chapter 4).

Structure of this thesis

This thesis is organized into four main chapters.

- **Chapter 1** gives an introduction to sequence alignment, methods of traditional sequencing and the arrival of high-throughput sequencing technologies. Furthermore, new computational challenges for mapping gigabytes of high-throughput reads are discussed as well current algorithms and software aimed at solving them.
- **Chapter 2** presents the new approximate seed and the supporting data structures used to quickly search for short regions of similarity between two sequences.
- **Chapter 3** presents the software SortMeRNA which implements the techniques of Chapter 2 to quickly and accurately filter rRNA fragments produced by high-throughput sequencing technologies.

- **Chapter 4** presents the software SortMeDNA for read mapping which extends the algorithm of SortMeRNA to perform full alignments of high-throughput genomic or metagenomic sequences. SortMeDNA also applies statistical analysis to evaluate the significance of each alignment.
- **Conclusion** summarizes our findings and gives perspectives for future works.

Chapter 1

Background to DNA sequence analysis

Contents

1.1 Overview

Biological sequence alignment is a method used to identify regions of similarity between organisms at the genomic, protein or structural level. Numerous applications include tracking species diversity and evolution, identifying new strains of bacteria or viruses, and predicting functional roles of organisms based on the similar genes they share. In genomic sequence alignment, the sequences used for comparison are derived from *deoxyribonucleic acid* (DNA) or *ribonucleic acid* (RNA) strands, which are long chains of nucleotides present in every cell of an organism. These chains are defined on the alphabet {A,C,G,T/U} and comprise both coding regions such as protein coding genes, and noncoding regions such as repetitive DNA, telomeres, and noncoding RNA. The most recurring form of RNA is a single, linear strand of nucleotides, whereas for DNA, it is a double stranded helix whose two strands run in opposite directions of one another. In protein sequence alignment, the sequences used for comparison are long chains of *amino acids*, frequently defined on a 20 letter alphabet. An amino acid is encoded by 3 nucleotides, collectively known as a *codon*, and some amino acids are encoded by more than one codon (ex. codons UCU and UCA both encode for amino acid Serine). *Sequence alignment* is the most commonly used method to compare two sequences by arranging them in a manner as to highlight their regions of similarity. Furthermore, secondary and tertiary structure information can complement sequence alignment to achieve a higher accuracy, such as reported for rRNA [Letsch et al., 2010] and protein molecules [Russell and Barton, 1992]. In this thesis, we focus exclusively on the sequence alignment between nucleotide sequences and apply our algorithms to DNA and RNA sequences generated by high-throughput sequencing technologies. In Section 1.2, we discuss the history and foundation of sequence alignment and demonstrate the fundamental algorithm behind most of today's alignment tools. In Section 1.3 we introduce Sanger sequencing, the pioneering 1^{st} generation sequencing technology still used for small-scale DNA projects such as small bacterial or viral genomes [Yan et al., 2013] or hypervariable regions such as in mitochondrial DNA [Lemay et al., 2013]. We then go on to describe some of the leading 2^{nd} and 3^{rd} generation sequencing technologies which are vastly applied for large-scale studies such as eukaryotic genome resequencing [Nystedt et al., 2013] and environmental genomics [Hingamp et al., 2013].

To this day, all sequencing technologies depend upon bioinformatic tools to post-process their sequenced data. These two fields crisscross in a manner that one's success will often lead to the other's. Improvements in the data generated by sequencing technologies, such as longer read lengths and reduced error-rates, correspondingly improve the performance and accuracy of existing sequence alignment tools. On the other hand, new improvements in software tools, such as better algorithms for error handling and faster speeds, stimulate research for novel methods of sequencing. In any respect, both research areas are continually evolving to provide the most effective and low-cost approaches to whole-genome analysis. In Section 1.4, we describe the leading bioinformatic tools used for mapping data issued by high-throughput sequencing technologies.

1.2 Sequence alignment

1.2.1 Foundation

Intuitively, short and closely related sequences, such as the phrases `lalunaebella` and `laluneestbelle` can be easily aligned by hand. In fact, they can be aligned in any of the following forms shown in Figure 1.1, using either the entire sequences for alignment or only their selected regions.

Figure 1.1: Global (1-2) and local (3-4) alignments for strings `lalunaebella` and `laluneestbelle`.

```
target   LALUNAE---BELLA     LALUNAE--BELLA     LALUNEE      LALUNAE--BELL
         |||*| |   ||||*     |||||*|  ||||*     |||||*|      |||||*|  ||||
query    LALUN-EESTBELLE     LALUNEESTBELLE     LALUNAE      LALUNEESTBELL
               (1)                 (2)             (3)              (4)
```

For any given alignment, one can associate mismatches (noted by `*`) to *substitutions* in biology, and gaps (noted by `-`) as *indel* (insertion and deletion) mutations. The regions where the sequences match exactly (noted by `|`) depict areas of high conservation. A *global alignment* will span the length of the entire target sequence (see Figure 1.1 (1),(2)), whereas a *local alignment* will only consider regions of high similarity between the sequences (see Figure 1.1 (3),(4)). If an alignment is found to be significant, it may represent a highly conserved region between two sequences which can help to compute their evolutionary distances or identify homologies based on, for example, overlapping sequence motifs [Kumar and Filipski, 2007]. Normally, for highly divergent sequences having undergone more complicated genome rearrangements, such as duplications and inversions of thousands of nucleotides [Lupski and Stankiewics, 2005], local alignment will be a better choice as it will localize isolated regions of similarity and will not expect them to have the same order and orientation on the sequences. If however the order and orientation of similarity regions are important, for instance during the construction of phylogenetic trees using metabolic pathways [Ma et al., 2013], then global alignment would be better-suited for the task. In practice, local alignment is a more popular approach due to its wide range of applicability and the fact that relatively few highly similar complete genomes or sequences are available to examine in full-length.

Apart from the type of alignment one must consider, how do we distinguish the best, or most biologically significant alignment? Logically, the correct alignment would be the one that captures the true changes made throughout time for a pair of sequences sharing a common ancestor. However, this is often not possible for nucleotide sequences, since these changes are introduced over long periods (up to millions of years) and depend on many different environmental and hereditary factors. Instead, we opt to search for an *optimal alignment* which maximizes the number of similarities between two sequences, and applies an alignment scoring scheme to do so. The scoring scheme assigns nucleotide matches, mismatches and gaps in an alignment with reward and penalty values, respectively, and at the end of an alignment the values are summed to yield an overall score. The alignment with the highest score is considered to be the optimal alignment. For protein alignments, it is customary to use log-odds matrices such as PAM [Dayhoff and Schwartz, 1978] or BLOSUM [Henikoff and Henikoff, 1992] to set the scores for substitutions, whereas the gap open and gap extension penalties are left for the users to decide, although techniques such as inverse parametric sequence alignment [Kim and Kececioglu, 2007] can be used to set these parameters based on biologically correct reference alignments. To a much lesser degree, few works have been published on the assessment of optimal alignment parameters for nucleotide sequences. In Subsection 1.2.3, we describe the few experiments made (based on observed evolutionary patterns) to address this question and give a general guideline for choosing appropriate parameters.

1.2.2 Dynamic programming for sequence alignment

Dynamic programming is a method of dividing complex problems into several smaller subproblems and solving each one individually, then combining the results to give a complete (and optimal) solution to the original problem. Often, this method will reduce the computation from exponential to polynomial time in a problem involving recursion, by effectively reducing the number of repetitive calls made to reach the final solution. In bioinformatics, dynamic programming is the method of choice for finding an alignment between two sequences. The first *global* sequence alignment algorithm, known as the Needleman-Wunsch [Needleman and Wunsch, 1970] algorithm, was developed in 1970 for aligning two proteins in full-length, although it can be equally applied to nucleotide sequences. This dynamic programming algorithm guarantees to find the correct optimal alignment between two sequences of lengths n in $O(n^2)$ time. A modified version of the algorithm by Smith & Waterman [Smith and Waterman, 1981] was introduced a decade later to search for an optimal *local* alignment. The root idea behind both algorithms is to explore the space of all alignments without having to actually list them, as that would require exponential time, by progressively computing the maximum scoring alignment along the lengths of the two sequences.

Under a *linear gap penalty model*, the cost of introducing a gap is independent of any previous gaps and it can be computed using a linear function $\gamma(l) = -ld$ where l is the length of the gap and d is the penalty score. The setup to this algorithm is a 2-dimensional $(n \times m)$ table representing all possible pairs of residues between the two sequences. For the Smith-Waterman algorithm, the entries in the alignment table M are computed using the rules listed in Figure 1.2.

Figure 1.2: Rules to construct the Smith-Waterman alignment table M of size $n \times m$ under a *linear gap penalty model*

$$(1)\ \ M(i,0) = 0, \forall i \in [0,m]$$

$$(2)\ \ M(0,j) = 0, \forall j \in [0,n]$$

$$(3)\ \ M(i,j) = max \begin{cases} 0 & \\ M(i-1,j-1) + w(x_i,y_j) & \text{if match } x_i \text{ with } y_i \\ M(i-1,j) + d & \text{if insertion in } x \\ M(i,j-1) + d & \text{if insertion in } y \end{cases}$$

$$\forall i \in [1,m], \forall j \in [1,n], d < 0$$

The function $w(x_i, y_i)$ usually returns a positive score for a match if $x_i = y_i$ or a negative score for a substitution if $x_i \neq y_i$. The parameter d is the score for opening a gap (ex. introducing an insertion or deletion with respect to the query sequence). For a guide to choosing values for the alignment scores (match, mismatch, insertion, deletion), see subsection 1.2.3. In the following example, we will use the values match = 2, mismatch = -3, gap = -5 and go on to align the sequences,

$$\text{target} = \texttt{ATAGCCTTT} \qquad \text{and} \qquad \text{query} = \texttt{ATCGCCTT}. \tag{1.1}$$

To begin with the table construction, we place the target sequence on the horizontal axis of the table and the query sequence on the vertical axis, as illustrated in Figure 1.3.

The first row and first column entries are all set to 0, in order to simulate an alignment of all letters with a null character, and the remainder of the computation begins at the first empty cell in the top left corner of the table. The table values in Figure 1.3 are computed in the left to right, top to bottom manner beginning from $M(1,1)$. At each new cell, the optimal score is updated using the previously computed values (see Figure 1.2 for the complete set of rules) and an arrow is placed from the current active cell to the cell used in the computation (the green arrow in Figure 1.3 signifies the value '1' was used to obtain the score, such that $M(4,4) = M(3,3) + 2 = 1 + 2 = 3$). Once the table has been filled in, the arrows are used to trace back a path for an alignment. The optimal local alignment begins from the highest computed value in the table, being the value 11 in Figure 1.4. To output the optimal alignment, we follow the arrows until we reach the end of the path. For our example, the optimal scoring path between the two sequences in our example is $11 \rightarrow 9 \rightarrow 7 \rightarrow 5 \rightarrow 3 \rightarrow 1 \rightarrow 4 \rightarrow 2$, which yields the local alignment shown in Figure 1.5.

Under an *affine gap penalty model*, the location of gaps relative to one another is considered since in biology gaps spanning multiple residues (multiple base insertions or deletions) are more likely to occur than as single residue mutations. A more fitting gap penalty for this property

Figure 1.3: Construction of Smith-Waterman table using dynamic programming

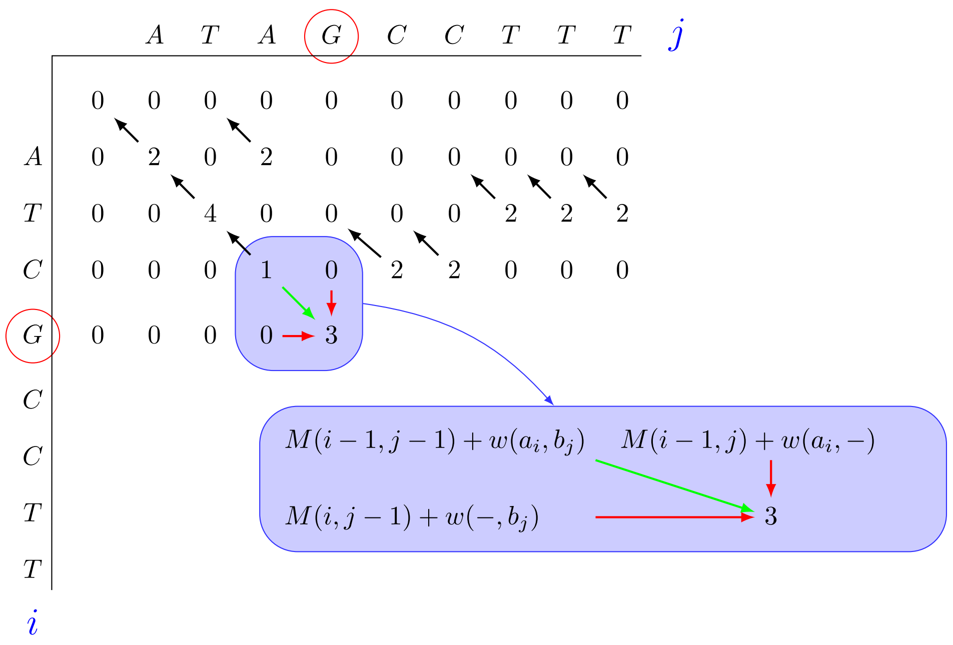
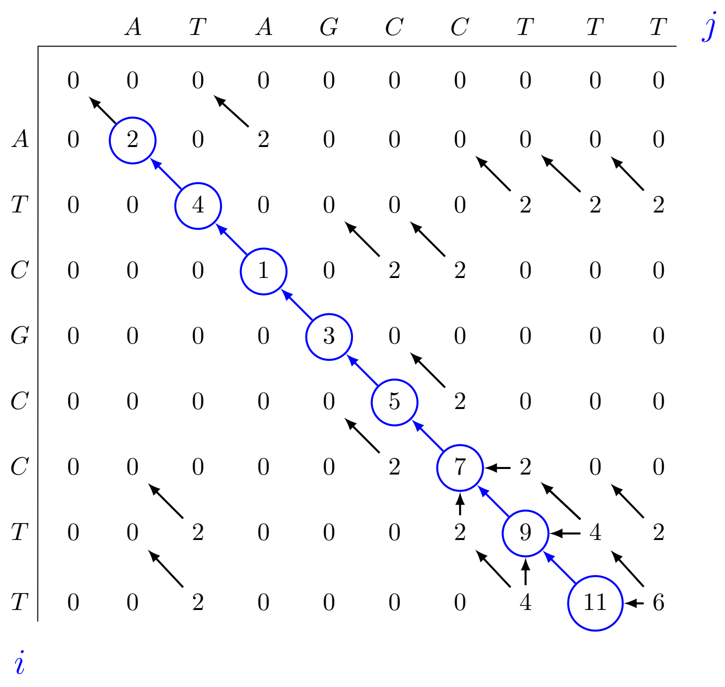


Figure 1.4: Optimal alignment trace back using Smith-Waterman table



follows the form $\gamma(l) = d + (l-1)e$ for $l \geq 1$, where l is the length of the gap, d is the score for opening a gap and e is the score for extending a gap such that $d > e$. In this setup, additional gaps are penalized less than the initial ones. The recurrence relation shown in Figure 1.2 can be

Figure 1.5: Optimal local alignment for sequences `ATAGCCTTT` and `ATCGCCTT` using the Smith-Waterman algorithm.

```
target     ATAGCCTT
           ||*|||||
query      ATCGCCTT
```

modified to the one shown in Figure 1.6. Now, instead of keeping track only of the best score given that x_i is aligned with y_j, we must also keep track of two other tables: I_x and I_y. The table I_x keeps track of the best score given that x_i is aligned with a gap, and similarly the table I_y keeps track of the best score given that y_j is aligned with a gap. The recurrence relation in Figure 1.6 can be represented as a finite state automaton (FSA) shown in Figure 1.7. For every cell $M(i,j)$ the FSA is traversed to a new state $(M, I_x$ or $I_y)$ from the previous one depending on the next pair of residues (x_i, y_j) used for the alignment. Similar to the linear gap penalty model, the time required to align two sequences is $O(n^2)$. Ideally, the gap penalty model should follow a concave-like curve such that every new additional gap receives a slightly lower penalty than the previous one. However, the dynamic programming algorithm would require more time to align two sequences since for each new cell (x_i, y_j) we must consider every previous cell in the column *and* every previous cell in the row (extra $i + j + 1$ computations), and not only the one previous computation. Therefore, the affine gap penalty model is the most practiced in biological sequence alignment and can be implemented using the algorithm described in [Gotoh, 1982].

Figure 1.6: Rules to construct the Smith-Waterman alignment table M of size $n \times m$ under an *affine gap penalty model*

$$(1)\ \ M(i,0) = 0, I_x(i,0) = -\infty, I_y(i,0) = -\infty\ \forall i \in [0,m]$$

$$(2)\ \ M(0,j) = 0, I_x(0,j) = -\infty, I_y(0,j) = -\infty\ \forall j \in [0,n]$$

$$(3)\ \ M(i,j) = max \begin{cases} 0 \\ M(i-1,j-1) + w(x_i,y_j) & \text{if match } x_i \text{ with } y_i \\ I_x(i-1,j-1) + w(x_i,y_i) & \text{if insertion in } x \\ I_y(i-1,j-1) + w(x_i,y_i) & \text{if insertion in } y \end{cases}$$

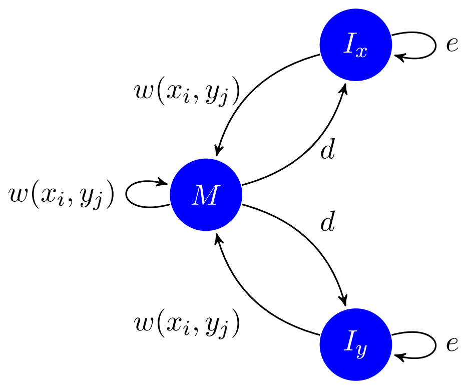
$$I_x(i,j) = max \begin{cases} M(i-1,j) + d & \text{if open gap in } x \\ I_x(i-1,j) + e & \text{if extend gap in } x \end{cases}$$

$$I_y(i,j) = max \begin{cases} M(i,j-1) + d & \text{if open gap in } y \\ I_y(i,j-1) + e & \text{if extend gap in } y \end{cases}$$

$$\forall i \in [1,m], \forall j \in [1,n], d < 0, e < 0$$

Although dynamic programming offers an important speedup over the naive recursion approach, it remains slow for aligning large eukaryotic genomes and when searching a large file of reads against a large database. To resolve this problem, significant efforts have been dedicated to improve the algorithm's performance using single instruction, multiple data (SIMD) parallelism. This accelerated approach has been reported to gain more than six times the speed over the traditional one [Farrar, 2007, Rognes, 2011] and has been implemented into many modern alignment software such as Novoalign (Novocraft), SHRiMP2 [David et al., 2011] and

Figure 1.7: A diagram of the relationships between the three states used for affine gap alignment [Durbin et al., 1998] (note $d < 0$ and $e < 0$)



Bowtie2 [Langmead and Salzberg, 2012]. Another acceleration used by all these tools is that none of them apply the dynamic programming algorithm to the entire search space of two sequences, but only to subregions showing potential to generate a high scoring alignment. In Section 1.4 we discuss the common heuristics used by many current read mapping tools and the efficacy of these approaches for different alignment applications.

1.2.3 Choosing alignment parameters for nucleotide sequences

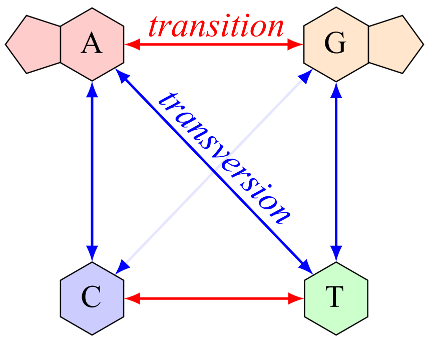
The Smith-Waterman sequence alignment parameters aim to maximize the score for true homologies of highly similar or distantly related species. Frequently, two different sets of parameters will produce two different results, especially if the query and reference data experience high rates of divergence. Since the start of the new millennium, very few investigations have been made into understanding the behavior of different alignment parameters for varying sets of data [Chiaromonte et al., 2002, Frith et al., 2010]. However, biological evolution suggests that substitution mutations occur more frequently than indels [Saitou and Ueda, 1994, Iengar, 2012], and thus should have a lower penalty in the scoring scheme. As a rule of thumb, a match should have a positive score and the mismatch, gap open and gap extend parameters should have negative scores such that the expected overall score for aligning two random sequences is negative. A simple model for a nucleotide scoring scheme is match=1, mismatch=-1, gap open=-2 and gap extend=-1. Interestingly, even though this model does not take into consideration any relationships between nucleotide bonds or mutation biases, it has been shown to work reasonably well on various data [Frith et al., 2010]. Indeed, more sophisticated models can be developed based on nucleotide compositions of known species to help increase accuracy of the results [States et al., 1991]. Certain types of models begin to consider biological and statistical properties of substitution mutations, as these are subject to certain biases in some species and do not come about as completely random events. For example, one of the species known to cause severe malaria in humans, the *Plasmodium falciparum*, has shown to experience a significant increase in mutations for two genes associated to sulfadoxine-pyrimethamine (drug) resistance [Ahmed et al., 2004].

Some of the commonly studied biases include CpG density, G+C content and transition/transversion ratios. A CpG site is a region of DNA where a C nucleotide is followed by a G nucleotide, or *vice versa*, along the same strand of DNA (this is different from a base-pair).

The CpG density shows a bias when the observed number of CpG dinucleotide sites does not correlate with the expected number (based on the G+C content), as witnessed in the subfamilies of HIV [Kypr et al., 1989] and the human genome [Sved and Bird, 1990]. A scoring scheme may be refined for acknowledging the presence of CpG dinucleotides by giving lower mismatch scores to more probable mutations of adjacent nucleotides, such as CpG$\rightarrow$TpG (or complementary CpA), a common mutation caused by the methylation of cytosine and its subsequent deamination into thymine [Jabbari and Bernardi, 2004]. It has also been shown that mutations of CpG sites can affect the mutation rate of non-CpG DNA [Walser et al., 2008], a possible aftermath of incorrect DNA reparation of the deaminated nucleotide. However, the latter observation is largely specific to the sequence in question and would require strong knowledge of its evolutionary origins in order to be scored effectively. Another often recognized bias is related to the G+C (or A+T) content, which is simply the skewed percentage of nucleotides being either G or C in a genome. The G+C content can vary significantly between different genomes, such as the G+C rich actinobacteria or A+T rich firmicutes, and this characteristic can be incorporated into a scoring matrix by assigning higher scores to matches between less frequent residue pairs (eg. if G+C rich, assign higher scores to matches between A-A and T-T and lower scores to matches between G-G and C-C [Frith et al., 2010]). Lastly, the transition/transversion bias refers to the greater likelihood of seeing a mutation from a purine to a purine (A$\leftrightarrow$G), or a pyrimidine to a pyrimidine (C$\leftrightarrow$T) nucleotide, than from the purine to a pyrimidine or *vice versa*, the possibilities are illustrated in Figure 1.8. The expected ratio for a transition to transversion mutation is 1:2, however in vertebrate genomes transition mutations are more likely to occur than transversions (driven by methylation) [Zhao and Boerwinkle, 2002], thus a scoring matrix can be adjusted to consider this property by giving higher scores for a transition (i.e. A$\rightarrow$G) mutation than for a transversion (i.e. A$\rightarrow$T) mutation. However, it was also shown in [Yang and Yoder, 1999] that for a group of eutherian species the transition-transversion ratio approaches 1 for levels of divergence $\geq 20\%$ due to a saturation of transitions. Investigations into using *sequence specific* transition-transversion substitution matrices have been presented using the tool DNAlignTT [Agrawal and Huang, 2008], which reported comparable and sometimes significantly better results over the trivial match/mismatch scoring scheme in their preliminary results.

Essentially, the substitution scores of a scoring matrix are derived from log-odds ratios [Karlin and Altschul, 1990], which compare the probabilities of aligning two bases as a result of evolution versus as a result of a random alignment. The biases discussed in this section can be used to construct a simple 4×4 matrix, being the safest choice for data with unknown properties, or a 16×16 matrix if considering dinucleotide properties [Salama and Stekel, 2013]. For coding regions, an amino acid scoring matrix can be used which aligns amino acids rather than individual nucleotides. Along with an appropriate scoring scheme, it is important to choose an alignment cutoff score, a score which will capture enough biological homologies but also dismiss artificial alignments which happen to arise by chance. A good measure of alignment significance is the expectation value (E-value) which gives the expected number of alignments between two random sequences of lengths m and n having a certain alignment score. In the following Subsection 1.2.4, we will discuss the E-value and its application in our software SortMeDNA, further described in Chapter 4.

Figure 1.8: Transition and transversion mutations



1.2.4 Significance of alignment scores

The purchasing power of a dollar can define its relative value, so what then defines the relative value of a sequence alignment? Well, we can begin by asking how many random sequence alignments can statistically exist that achieve the same score as the homology at question. In [Gordon et al., 1986, Karlin and Altschul, 1990] it was proven that the distribution of the Smith-Waterman ungapped local alignment scores between two infinitely long random sequences (each successive character follows a Markovian model) follow the extreme value distribution type I, otherwise known as the Gumbel distribution. Moreover, the latter paper goes on to describe the expectation value (E-value), that is the number of random alignments expected to exist between two sequences of lengths m and n having a score S or greater. The equation for this calculation is given by,

$$E = Kmne^{-\lambda S} \tag{1.2}$$

where K and λ are the Gumbel parameters, m and n are the lengths of the query and reference sequences, respectively, and S is their Smith-Waterman local alignment score. The logic behind Equation 1.2 follows the same principle as observing the longest run of heads (or tails) in a game of coin toss. During an alignment between two random sequences, we can consider a match between two letters as throwing a head and a mismatch as throwing a tail. Because of the randomness property, every subsequent comparison between two letters will yield a result independent from any previous one, synonymous to each new toss of an unbiased coin.

Although it has not been proven that an alignment involving gaps also follows the Gumbel distribution, it has been heavily conjectured to be the case through simulation analysis [Altschul and Gish, 1996]. The Gumbel parameters K and λ are the scaling factors and are computed using the background nucleotide frequencies of the reference sequence n and an alignment scoring matrix (including indel costs). The most straightforward approach to compute these values is to simulate many random sequences of lengths n and m, align them and fit the values K and λ onto the resulting score distribution [Waterman and Vingron, 1994]. However, this approach is inefficient since it requires thousands of random alignments to arrive at an accurate estimation. Various faster and on-the-fly methods have been developed to alleviate this time constraint by using the 'islands method' [Olsen et al., 1999], global alignment [Sheetlin et al., 2005] or importance sampling [Park et al., 2009] techniques.

The first, and still one of the few, local alignment tools that effectively applies this statistical measure for evaluating alignment score significance is BLAST [Altschul et al., 1990]. BLAST computes the E-value per query, meaning that the length m is for one query and n for the entire reference database. It is also possible to compute the E-value for the entire search space, meaning the length m will represent all of query sequences and n, as in BLAST, will represent the entire reference database. When computing the E-value, it is also important to consider alignments carried out on the edges of either the query or reference sequences, where an alignment may run out of sequence before reaching the threshold score. To account for this boundary condition, we also compute the finite-size correction which adjusts the lengths of m and n by subtracting from them the minimal length l required to achieve a significant score. When computing the length correction per read, the relative entropy is used to take into account the nucleotide distributions for both the query and reference sequence. However, when computing it for the entire set of reads, the entropy of the reference sequence alone can be used. Although this approach is not theoretically optimal [Wang and Samudrala, 2006], it allows to compute the length correction one time and apply it to the global computation of the E-value for all reads. Below we outline the steps taken to compute the E-value for the entire search space, which apart from approximating the length correction using entropy (rather than relative entropy), closely follow those for BLASTN [Korf et al., 2003].

Let M and N define the total number of query and reference sequences in our search space, respectively. Then, we can represent the m and n values in Equation 1.2 as,

$$m = \sum_{i=1}^{M} m_i \qquad\qquad n = \sum_{i=1}^{N} n_i \tag{1.3}$$

Considering the finite-size correction, we must additionally subtract this value l from every query and reference sequence,

$$m' = m - (l \times M) \qquad\qquad n' = n - (l \times N) \tag{1.4}$$

The length correction l is given by,

$$l = \frac{\ln(Kmn)}{H} \tag{1.5}$$

If $l > m$ then set $m' = \frac{1}{k}$. The parameter H represents the relative Shannon entropy,

$$H = -\sum_{i=1}^{4}\sum_{j=1}^{i} q_{ij} \ln \frac{q_{ij}}{p_i p_j} \tag{1.6}$$

for which q_{ij} are the pair-wise frequencies of the aligned region and p_i, p_j are the background frequencies of nucleotides in the reference database. However, since we would like to compute the E-value for all query sequences rather than per query, we assume that the pair-wise frequency probabilities are unknown or resemble the background frequencies of the reference database (see [Wang and Samudrala, 2006]). We use the Shannon entropy instead,

$$H = -\sum_{i=1}^{4} p_i \ln p_i \tag{1.7}$$

Therefore, the E-value with length correction for m and n is,

$$E = Km'n'e^{-\lambda S} \tag{1.8}$$

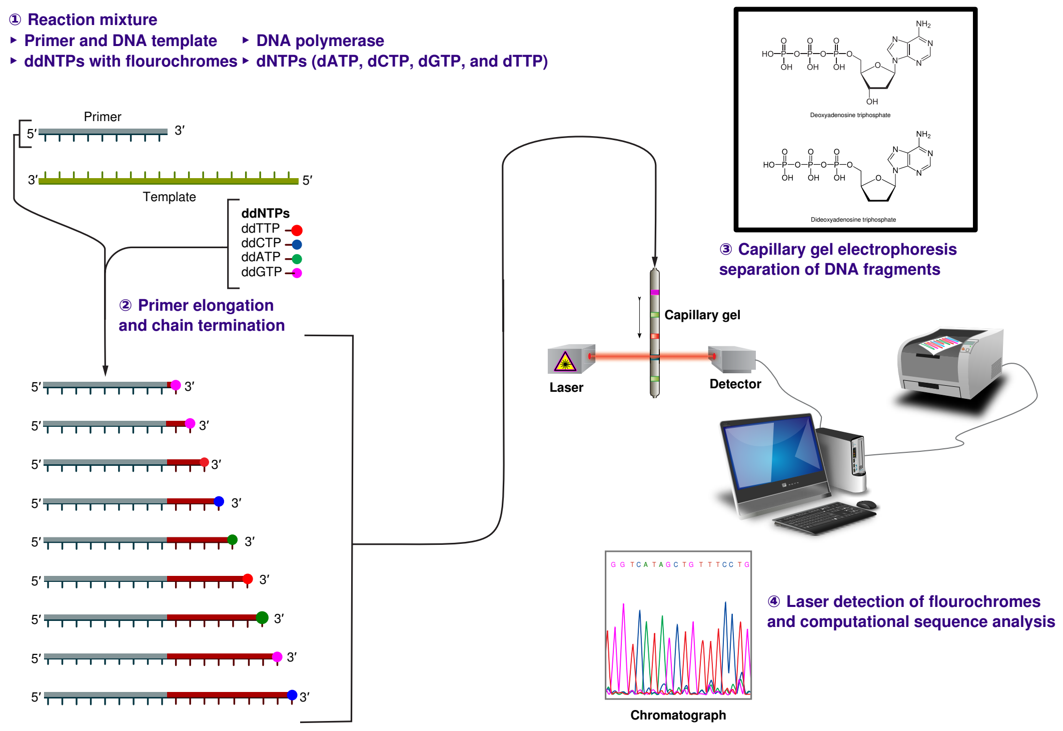
1.3 Next generation sequencing

1.3.1 1st generation Sanger sequencing

The first complete genome, the ϕX174 virus, was sequenced in 1977 [Sanger et al., 1977] using a *chain termination technique* developed by Frederick Sanger. The original method required four reaction mixtures to be set up each containing single-stranded copies of a DNA template, DNA polymerase enzymes responsible for DNA replication, DNA primers (short DNA sequences complementary to the 5' end of the template sequence) and equal amounts of all four types of nucleotides. In addition, small amounts of *one type* of modified nucleotide are added to each reaction mixture. The modified nucleotides lack the hydroxyl group (in the sugar molecule) required for bonding of adjacent nucleotides during replication. Depending on the variant preferred, either the normal nucleotides, the modified nucleotides or the DNA primers are also radioactively labeled in order to see the DNA molecule in the final step.

The chain termination method begins with a single-stranded DNA template to which a primer is annealed at the 5' end (DNA can only be synthesized from the 5'→3' direction). As the DNA polymerase gradually incorporates normal nucleotides to form the complementary strand, the synthesis reaction proceeds until a modified nucleotide is integrated into the chain, terminating elongation and essentially exposing which of A,C,G or T nucleotides resides at that position. Therefore, in order to determine the position of every single nucleotide in a strand of DNA, this method must be applied to many copies of the same strand (in each reaction mixture) obtainable via plasmid vectors or the polymerase chain reaction (PCR). The final step of sequencing involves denaturing the elongated DNA fragments from their template strands and passing them through a gel electrophoresis setup (different lane for each reaction mixture), which separates the fragments according to length differing in size by 1 bp. Since each fragment is radioactively labeled (via primer, normal nucleotides or modified nucleotide), all that remains to do is read the radioactive bands from the start 5'→3' in the four lanes of gel to determine the complete sequence of the original DNA strand. In 1986, an automated variation to the Sanger method was described [Smith et al., 1986] where the modified nucleotides are fluorescently labeled with a different color to distinguish each type and added simultaneously into one reaction mixture. Afterwards, during electrophoresis a laser excites the atoms in the fluorescent labels to produce visible light which is translated by a computer into its corresponding nucleotide. Refer to Figure 1.9 for an illustration of the automated method. The main limitation to this approach of sequencing is that it can only be applied to short DNA sequences (∼800 bp back then and ∼2,645 bp today using Novex® precast gels by Life Technologies), in order to maintain a high resolution during gel electrophoresis for a pair of sequences differing in length by 1 bp. Therefore, even a genome as small as the one of the ϕX174 virus had to be sequenced using a *whole-genome shotgun* approach, where the entire molecule was randomly sheared into smaller, possibly overlapping DNA fragments (∼500 bp) to be individually sequenced using the chain termination method and finally reassembled back into the original genome using computational approaches [Staden, 1979].

Figure 1.9: Sanger sequencing using the chain termination method, automated version using fluorescently labeled nucleotides and computer reading



① Reaction mixture
‣ Primer and DNA template
‣ DNA polymerase
‣ ddNTPs with flourochromes
‣ dNTPs (dATP, dCTP, dGTP, and dTTP)
Primer
Template
ddNTPs
ddTTP
ddCTP
ddATP
ddGTP
② Primer elongation and chain termination
③ Capillary gel electrophoresis separation of DNA fragments
Capillary gel
Laser
Detector
Deoxyadenosine triphosphate
Dideoxyadenosine triphosphate
④ Laser detection of flourochromes and computational sequence analysis
Chromatograph

http://commons.wikimedia.org/wiki/File:Sanger-sequencing.svg

Milestones using Sanger sequencing

The first bacterial genome, *Haemophilus influenzae*, was entirely sequenced using the whole-genome shotgun sequencing approach and afterwards reassembled with the TIGR assembler [Fleischmann et al., 1995]. The sequencing of the first eukaryotic genome, the yeast *Saccharomyces cerevisiae*, was an international collaboration effort that commenced in 1989 and was completed seven years later [Goffeau et al., 1996]. Due to large computational resources required for assembling many small DNA fragments (and concerns of the authenticity of the results), the sequencing of *S. cerevisiae* was done in a hierarchical approach. Complete chromosomes were initially split up into ordered and slightly overlapping sections using BAC clones, and each section was then individually shotgunned into smaller fragments which were finally sequenced and reassembled back into the original parent sections. The Human Genome Project (HGP) was another international collaboration to sequence the first human genome using Sanger sequencing, beginning in 1990 and completing 13 years later with an estimated cost of $2.7 billion. The same hierarchical shotgun sequencing approach (as for *S. cerevisiae*) was used to sequence the 3 billion base-pair genome [Consortium, 2001]. Although a remarkable accomplishment for mankind, at that rate and cost personalized genome sequencing was unrealistic for large-scale and commercial applications. During the same time period, another team took on the challenge to sequence the human genome using whole-genome shotgun sequencing [Venter et al., 2001], largely omitting the map-based (BAC clones) sequencing steps. Localized regional shotgun sequencing was also performed to improve resolution by first sorting reads to known BAC contigs (derived from the

HGP) and then individually assembling them.

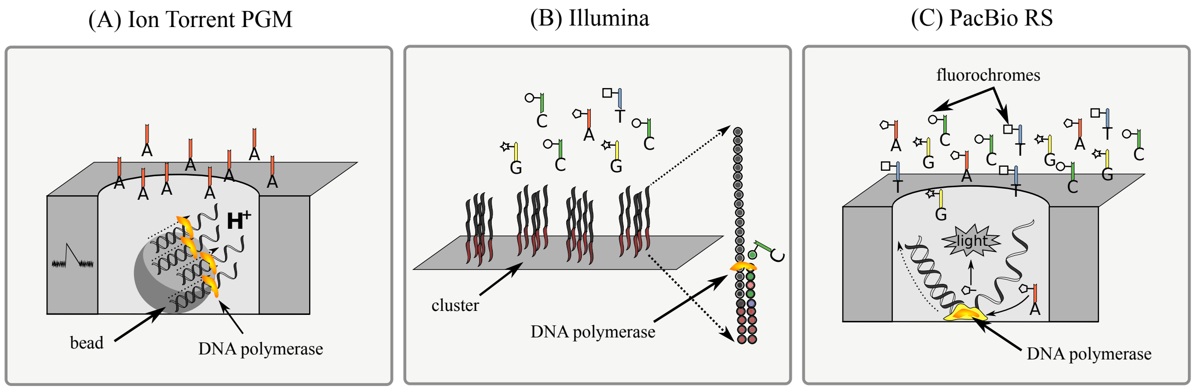
1.3.2 2nd and 3rd generation sequencing

Although the Sanger method can produce average read lengths of 850 bp having 99% consensus accuracy, the thoughput per run is less than 24 Kb (for a modern 24-capillary 3500xL Genetic Analyzer system) and can take up to 6 hours to prepare and run. The new millenium has brought forth many new state-of-the-art technologies which can sequence the entire human genome within two days under $10,000 (rapid run mode using the Illumina HiSeq 2500 system), allbeit often with significantly shorter read lengths and a higher error rate (at the time of writing).

Sequencing-by-synthesis is a technique used to sequence a single strand of DNA in real-time, by incorporating free nucleotides into a growing chain using DNA polymerase and immediately recording which nucleotide was added. In 1996, a sequencing-by-synthesis setup was used to sequence 15 bases of a 291 bp DNA template [Ronaghi et al., 1996] in real-time, setting a world record for this alternative approach to Sanger sequencing. Today, this cutting-edge principle is the foundation of all high-throughput sequencing technologies, such as 454 (Roche), Illumina (Solexa) and Ion Torrent (Life Technologies), adept for generating millions of high quality reads of lengths 100-1000 bp within several days. Similarly to Sanger sequencing, all sequencing-by-synthesis technologies employ DNA polymerase to gradually incorporate free nucleotides into a growing strand of complementary DNA. Per contra, the detection of incorporated nucleotides is performed using variant techniques other than the Sanger method and the entire sequencing process is massively parallelized. For 454, the method of detection is pyrosequencing [Ronaghi et al., 1998, Nyrén, 2007], for Ion Torrent it is the detection of hydrogen ions [Rothberg et al., 2011] and for Illumina it is nucleotide dye-termination [Bentley et al., 2008]. All of these techniques are performed over millions of amplified clones (derived via emulsion PCR using beads or clusters) for DNA fragments obtained via whole-genome shotgun sequencing, in order to provide a sufficient signal for detecting nucleotide incorporation. Moreover, the flow cells on which sequencing reactions take place, are optimized to permit massively parallel sequencing of millions of different DNA fragments simultaneously. The diagrams (A) and (B) in Figure 1.10 illustrate the general setup for Ion Torrent and Illumina platforms. In the following paragraphs we will give a short summary of each technique and highlight the steps most susceptible to *sequencing errors* and the factors limiting the production of long reads, as error types and read lengths vary between technologies (see Table 1.1) and are important to consider during the development of software for sequence analysis.

454 and Ion Torrent sequencing are based on detecting by-products of a synthesis reaction. During the synthesis of a DNA molecule, free nucleotides are incorporated in their natural triphosphate forms in cycles over a sequencing flow cell. Upon bonding of a free nucleotide to the next unpaired base in the template strand, two by-products, a pyrophosphate (two phosphate groups bonded together) and a positively charged hydrogen ion, are released. Each by-product is exclusively used by the 454 and Ion Torrent technologies, respectively, to detect the incorporation of a base. In the case of 454, the release of a high-energy pyrophosphate molecule triggers a series of catalytic reactions which ultimately end with an emission of visible light. The intensity of light emitted is proportional to the number of nucleotides incorporated into the DNA chain and can be detected by a charge-coupled device (CCD) camera. In the case of Ion Torrent, the release of a positively charged hydrogen ion decreases the pH of the solution (since pH $= -log_{10}(H^{+})$), and the change in ion concentration can be detected using a silicon substrate chip (see Figure 1.10(A)). In both cases, the addition of different types of nucleotides

Figure 1.10: Second and third generation sequencing technologies



Ion Torrent and PacBio images taken from http://en.wikipedia.org/wiki/File:From_second_to_fourth-generation_sequencing,_illustration_on_TAGGCT_template.svg, Illumina individually incorporated

is performed sequentially, in order to recognize which nucleotide was incorporated during the detection of a light signal or the change in ion concentration. One limitation to both methods is a higher error rate for detecting long homopolymers (a run of identical bases), which is caused by an over- or under-call of the signal for the incorporated nucleotides. These types of errors often manifest themselves as insertions or deletions of nucleotides and can sometimes lead to incorrect estimations of biodiversity for environmental studies [Fonseca et al., 2010]. Whether a nucleotide was incorporated or not, the flow cell must be thoroughly washed over to remove all unincorporated nucleotides before new enzymes and regents can be added for the next cycle (this will ensure synchronized signals and reduce byproduct accumulation which is known to inhibit DNA polymerase [Mashayekhi and Ronaghi, 2007]).

Illumina sequencing uses fluorescently labeled nucleotides (different ‘color’ for each type) which work to serve two purposes: firstly, the label prohibits further elongation of the complementary strand such that a single nucleotide can be incorporated at each flow cycle (this avoids the over- or under-call of bases as in 454 and Ion Torrent) and secondly, its proper color allows for the identification of the incorporated nucleotide (here a miscall of a base can occur leading to a substitution error). Illumina’s labeled nucleotides have the property of having reversible terminators, meaning that when the incorporated nucleotide is recorded by a camera based on its color, an enzyme can cleave off the fluorescent label to reinstituate synthesis. Because sequencing is carried out in a base-by-base manner, this technology produces far fewer errors at homopolymer and repetitive sequence regions than both 454 and Ion Torrent.

All of the three aforementioned sequencing approaches have limitations to the read lengths produced (see Table 1.1), some of these limiting factors attribute to nucleotide misincorporation by DNA polymerase or a reduced signal due to lost DNA fragments ($\sim$0.1%) washed away during each cycle [Mashayekhi and Ronaghi, 2007], as well as the complex management of massively parallelized setups. In terms of sequencing errors, it has been estimated that nowadays the Illumina technology has the lowest error rates in one round of sequencing at 0.1 errors per 100 bases, in comparison to 0.38 and 1.5 errors per 100 bases for 454 and Ion Torrent, respectively [Loman et al., 2012]. Error types occur mostly as substitutions for Illumina (frequently sequence-specific for GGC motifs and inverted repeats [Nakamura et al., 2011]) and homopolymer indels for 454 and Ion Torrent.

PacBio single-molecule sequencing is a recent[1] technology capable of generating average read lengths of 4600 bp (possible up to 24Kb). This advanced technology uses fluorescent labels attached to the terminal phosphate group of free nucleotides (as opposed to the actual base) which are cleaved off by DNA polymerase during replication (see Figure 1.10(C)). The single strand of DNA to be sequenced is passed through a single DNA polymerase molecule attached to a *zero-mode waveguide* (ZMW) at the bottom of a chamber. The ZMW is a structure sensitive enough to detect the incorporation of single nucleotides [Levene et al., 2003]. During sequencing, all four types of nucleotides are introduced into the chamber simultaneously at equal amounts. When one of the four nucleotides is being incorporated into the chain, its distinct fluorescent label excites for several milliseconds and this illumination signal is recorded by the ZMW. Thusly, the entire strand of DNA can be sequenced continuously using a single strand of DNA without the need for PCR amplification as for other technologies (known to cause sequence-specific biased coverage [Ross et al., 2013]), or cyclic washing of flow cells. Although the error rates are much higher than for other technologies, nearly 15% (mostly indels), they appear to be independent of the sequencing context and can be resolved by multiple rounds of resequencing [Roberts et al., 2013]. Of all today's technologies, PacBio delivers the least invasive method for eavesdroping on natural DNA replication while delivering very long reads in a short time period (see Table 1.1). Small bacterial genomes are already being fully sequenced and assembled using this technology [Nicholsona et al., 2013, Khosravi et al., 2013], since the long reads can cover complete gene transcripts and low complexity regions.

1.4 Tools for high-throughput read mapping

The arrival of high-throughput sequencing technologies has introduced new problems for read mapping. Firstly, the low cost of sequencing has allowed many smaller laboratories worldwide to invest in personal machines and sequence their own data. However, not every laboratory has the sufficient computational resources capable of analyzing such large amount of throughput data. Secondly, all of today's technologies are susceptible to sequencing errors in the form of substitutions, insertions and deletions (see Section 1.3.2). Moreover, naturally occurring errors may exist in distant homologies and being able to consider them is an important characteristic for aligning divergent species.

In the context of speed, SSEARCH [Smith and Waterman, 1981, Pearson, 1991] is a tool which provides a modern implementation of the exhaustive Smith-Waterman sequence alignment, however to align 1000 Illumina reads of length 100 nt against human chromosome 21 takes about a half-hour (using one thread on an Intel Xeon CPU @ 2.67GHz). Given that a typical Illumina MiSeq machine can generate 15 million reads within 2 days (see Figure 1.1), using SSEARCH to map all of these reads against the same chromosome would take roughly 1 year. New heuristics have been developed to accelerate this search by first locating short regions of similarity called *seeds* between the query and reference and then extending the seeds into longer alignments using the Smith-Waterman algorithm (see Figure 1.11). In other words, a seed gives a clue for a potential alignment. To highlight the speedup gained with heuristic approaches in regard to the full dynamic programming algorithm, Bowtie2 [Langmead and Salzberg, 2012] takes less than a minute to index human chromosome 21 and only about 0.2 seconds to align the same set of 1000

[1](11/04/2013, GlobeNewswire) Photo Release – Pacific Biosciences Launches the PacBio(R) RS II Sequencing System

Table 1.1: Specifications for various sequencing-by-synthesis technologies (taken from each company's official website 09/2013)

Platform	Read length (bp)	Reads per run	Run time	Base call accuracy	Application
2nd generation					
Illumina[1]					
MiSeq (paired-end)	250x2	15 million	39 hrs	99.9% for >75% of bases	Targeted, genome, metagenomics, transcriptomics, epigenetics
HiSeq 2500 (high output, paired-end)	100x2	6 billion	11 days	99.9% for >80% of bases	(same as MiSeq)
Roche/454[2]					
GS Junior	∼400	∼100,000	10 hrs	99% at first 400 bases	Targeted, genome, metagenomics, transcriptomics
GS FLX XL+	up to 1000 (average 700)	∼1,000,000	23 hrs	99.997% consensus at 15x coverage	(same as GS Junior)
Ion Torrent[3]					
Ion PGM (Ion 318 Chip v2)	200-400	4-5.5 million	4.4-7.3 hrs	99% expected	Exome, methylation, genome, transcriptomics
Ion Proton	up to 200	60-80 million	2-4 hrs	99% expected	(same as Ion PGM)
3rd generation					
Pacific BioScience[4]					
PacBio RS II	up to 23,297 (average 4,606)	47,197	30 min - 2 hrs	99.999% consensus at 30x coverage	*de novo* assembly, targeted

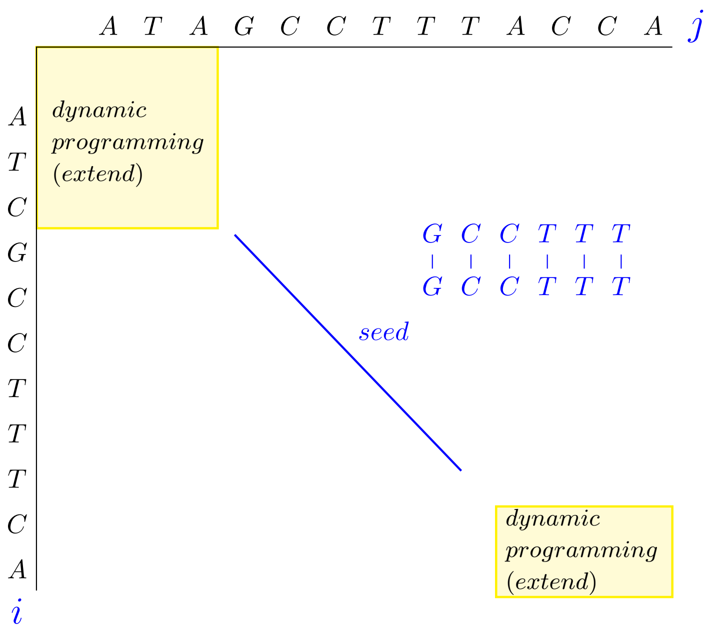
[1] www.illumina.com

[2] www.454.com

[3] www.lifetechnologies.com

[4] www.pacificbiosciences.com

Figure 1.11: Seed-and-extend strategy to reduce the amount of search space examined by the dynamic programming algorithm for aligning sequences x and y


A T A G C C T T T A C C A j
A T C G C C T T T C A i
dynamic programming (extend)
G C C T T T
G C C T T T
seed
dynamic programming (extend)

Illumina reads (or 50 minutes for the entire 15 million reads).

In the context of errors, many existing alignment tools have been optimized for genome resequencing (~99% sequence similarity) and adopt an exact match seed model. However, for applications such as metagenomics or metatranscriptomics where the reference sequences may share ~75-98% similarity to the query, these tools are no longer sensitive enough.

Most of today's read mapping tools use a seed-and-extend approach coupled with a query and/or reference index to facilitate rapid alignment of HTS data or sequences to large-scale databases. In the following subsections we will describe the early techniques of using suffix trees and enhanced suffix arrays to align sequences using *exact contiguous seeds*, as well as the FM-index now used in many of the recent alignment tools for its near-linear time search and compressed structure [Chacón et al., 2013]. Furthermore, we will discuss the advantages of using hash tables for implementing *exact noncontiguous seeds*, that is seeds allowing mismatch errors, and the enhanced sensitivity achieved by these tools for aligning more distantly related species. Finally, we will outline our motivation for a new indexing structure implementing *approximate seeds* allowing mismatch and gap errors, for a further improvement in aligning sequences exhibiting high indel error rates or distant homologies.

1.4.1 Heuristic seed alignment

Typically there are three types of seeds that are continuously studied in pattern matching algorithms. The first type of seed we will call the *exact contiguous seed*, simply meaning the match must be contiguous and share no errors between the query and the reference sequence. All indexing data structures used today can easily support exact contiguous seeds, including the suffix tree, (enhanced) suffix array, the FM-index and hash tables which are all discussed in the following subsections. Although exact matching algorithms are very fast, it has been shown in [Ma et al., 2002] that *exact noncontiguous seeds* or *spaced seeds* which allow mismatches at some predefined positions can improve sensitivity and help recognize distant homologies (provided that not too many indels are present). A spaced seed is defined by its size and weight. For example, a spaced seed of size 16 and weight 12 searches for k-mers (matches of length k) of length 16 but where only 12 of the predefined positions require to have an exact match. The most efficient indexing data structure by now which can support multiple *predefined* mismatches in the seed is based on hash tables. Suffix trees, the enhanced suffix array and the FM-index allow for mismatches also (not predefined to any positions) but the underlying backtracking algorithm can often render their search algorithms considerably slower, especially for a larger number of mismatches. However, sometimes even exact noncontiguous seeds are not sensitive enough to spot distant homologies (ex. environmental studies), or insertion/deletion errors specific to sequencing technologies such as 454, Ion Torrent and PacBio. In these contexts, *approximate seeds* allowing mismatch and indel errors anywhere in the seed would serve as an optimal choice (but at some computational cost) and the few tools today that explicitly implement them are based on some variant of the suffix tree.

1.4.2 Indexing using hash tables

A common practice in biological sequence alignment is to find all *exact* occurrences of a k-mer in a fixed text of length n, which can often be large and repetitive such as the human genome or a ribosomal RNA database. A naïve approach would take $O((n-k+1)k)$ time by placing the k-mer at all possible positions $(n-k+1)$ in the text n. The fastest alternative to this problem requires only constant O(z) time where z is the number of occurrences of a k-mer, by indexing the text in a lookup table (or perfect hash table).

Shortly following the publication of the Smith-Waterman dynamic programming algorithm, a paper emerged introducing the idea of using a lookup table to efficiently index and search nucleotide sequences [Dumas and Ninio, 1982]. By translating each consecutive pair of overlapping nucleotides into a number (for example, AC = 15) and using it as a lookup index in a table[2] storing the original positions of each dinucleotide, the authors could quickly identify all positions of any dinucleotide within the original sequence. This method was later extended to exact overlapping k-mers [Wilbur and Lipman, 1983], being substrings of length k, and used to find regions of high similarity between two sequences prior to extending them to full alignments using dynamic programming. For example, by using a 2-bits-per-nucleotide encoding where A=00, C=01, G=10 and T=11, we can encode the string x = tatagata to a binary number x = 1100110010001100 and subsequently use the decimal form x = 52364 as an index to a table. FASTA [Pearson and Lipman, 1988] and BLAST [Altschul et al., 1990] (BLASTN is the nucleotide version to which we will refer to from now on) are two established families of programs which index every query independently using contiguous k-mers and maintain an auxiliary lookup table for a list of their occurrences (for nucleotide sequences). Afterwards, a reference database is scanned for k-mers in common with the query using the lookup table and the best matching regions are extended into a full alignment. In this manner, only small subparts of the entire $O(n^2)$ search space are actually aligned, thus gaining a lot of time (see Figure 1.11). Although this heuristic method cannot guarantee to find all local alignments in contrast to the full dynamic programming algorithm, it has shown to be a good approximation in many applications

[2]The index was used to search the last occurrence of a dinucleotide in the primary lookup table, however a secondary table was also required to search for all preceding positions.

[Lipman and Pearson, 1985, Pearson and Lipman, 1988, Lam et al., 2008].

Since both FASTA and BLASTN localize their indexing to individual queries and the size of the k-mers is relatively small (smaller k-mers = more hits in the database), both of these tools are subject to long processing times for large data. For this reason, recently developed HTS alignment tools focus on indexing the full set of queries (MAQ [Li et al., 2008]), or the full set of reference sequences (Novoalign [Novocraft], GNUMAP [Clement et al., 2010] and SHRiMP2 [David et al., 2011]), or both at the same time (BWA-SW [Li and Durbin, 2009] but using trees). Although searching for each k-mer occurrence of a query in a reference sequence can be done in constant time using this structure, the memory requirement to index the human genome is > 15Gb (for Novoalign, SHRiMP2 and BFAST [Homer et al., 2009]). BFAST applies a two-level hash system for long k-mers, where only the first $p < k$ nucleotides are hashed and the remaining parts are stored in a complementary bucket. During a lookup, the first p characters are used to index the hash table and then a binary search is performed on the tails in the complementary bucket, resulting in a lookup time slightly inferior to $O(1)$.

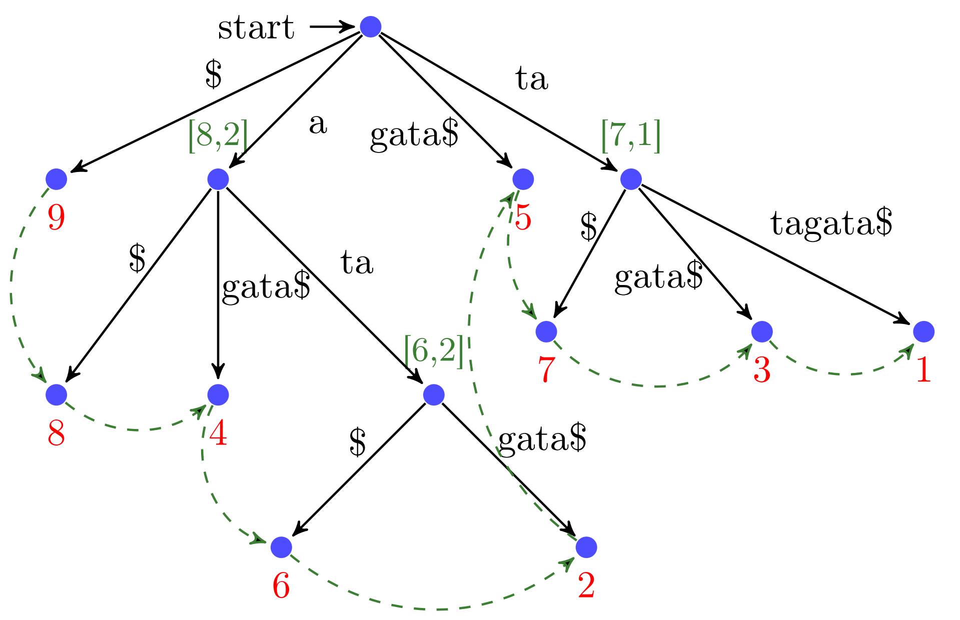
1.4.3 Indexing using suffix trees

An alternative to using a hash table to search for all k-mer occurrences is to use a *suffix tree*, which requires only $O(k+z)$ time (where z is the number of occurrences of the k-mer). One advantage of suffix trees over hash tables is that they can be used with a collection of strings of varying lengths and they also support approximate string matching [Navarro and Baeza-Yates, 2000, Russo et al., 2009]. Another advantage is that identical prefices of suffices are clustered into one path, thus greatly reducing the number of comparisons. Also, a tree traversal is rapid and can be performed in various modes to accommodate different applications [Gusfield, 1997]. The main drawback of a suffix tree is the large amount of memory it requires, with the most efficient implementation to date consuming 12-17 bytes per nucleotide [Kurtz, 1999, Kurtz et al., 2004]. For this reason, several more compact versions of the suffix tree have been developed, namely the enhanced suffix array and the suffix array-like FM-index. In the following subsections we will briefly describe these data structures and the software tools which implement them.

1.4.3.1 Suffix tree

Suffix trees have been used since the 1970's [Weiner, 1973, McCreight, 1976] to efficiently store all prefices or suffices of a string in a non-redundant manner. There exist both linear time methods for constructing a suffix tree [Ukkonen, 1995] on a constant-size alphabet (online-construction) and for any alphabet [Farach, 1997]. A suffix tree can represent all of the substrings of a string in an easily accessible format. For example, the suffix tree for the string x = tatagata\$ is shown in Figure 1.12. Every leaf node corresponds to a suffix and its label remarks the starting position of the suffix within the original string. Appending the special character '\$' to a string before constructing its suffix tree ensures that a preorder traversal yields the suffix nodes in lexicographical order, as '\$' is considered to be the lexicographically least character. To search a k-mer in the tree, one must begin at the root node (marked with 'start') and trace a path through the branches whose characters match exactly to the query. If all letters of a k-mer are exhausted during tree traversal at a non-leaf node j, then the number of occurrences of the k-mer in the string correspond to the number of terminal leaf nodes branching from j. To find all positions at which the k-mer occurs at, the suffix tree must also contain (a) a lexicographical linking between all leaf nodes (the dashed green arrows in Figure 1.12) (b) the paired information of the first and

Figure 1.12: Illustration of the suffix tree for the sequence x = tatagata\$. The leaf nodes hold the starting position (in color red) of each suffix in x. A preorder traversal of this suffix tree beginning from the root node (marked as start) yields all suffices of x in lexicographical order, being {\$,a\$,agata\$,ata\$,atagata\$,gata\$,ta\$,tagata\$,tatagata\$}. To search for all occurrences of a string, we begin at the root node and follow the edges that match to the characters of our string. The string (or at least its prefix) exists in the tree if we exhaust all of the characters before or at a leaf node. For example, if we search the string s = ata, we will finish at the inner node marked with [6,2]. The green dashed path links together all leaf nodes in lexicographical order and the [x,y] label at each inner node (except the root) gives the first and last position of a leaf node reachable from the current inner node. Both of these are optional as they are only useful for finding all of the locations at which s occurs (other methods exist too). To find all positions at which s occurs, we descend to the first lexicographically least leaf node and output its position (being 6). Then we follow the paths linking the leaf nodes and output their positions until we reach the last position (being 2).



last suffix position amongst all paths branching from j (the three green paired numbers [x,y] in Figure 1.12). In this manner, all occurrences of a k-mer and its positions in the original string can be recuperated. Otherwise, if at some node l no more outgoing branches match to the remaining unmatched characters in a k-mer, then the k-mer does not exist in the string.

OASIS [Meek et al., 2003] and MUMmer [Kurtz et al., 2004] are two sequence alignment tools that index the reference database using a suffix tree. Both of these tools require 12 to 17 bytes per nucleotide and are practical for whole-genome alignment of smaller bacterial genomes, as the resulting index can reside comfortably in a low-memory machine (>36 Gb required for the human genome) and maintain better cache locality. Another compromise to using a suffix tree-based aligner is that the index is often rebuilt for each new run, which can become time consuming for aligning large collections of high throughput reads. In a different application called RISOTTO [Pisanti et al., 2006], suffix trees were also used to quickly extract repeated patterns (functional elements in genetic sequences or entire genes) from a text allowing mismatch errors.

1.4.3.2 Suffix array

A *suffix array* (SA) is a sorted array of all suffices' starting positions in a string. It was first introduced in [Gonnet et al., 1992, Manber and Myers, 1993] as a space-efficient alternative to a suffix tree for finding all occurrences of a pattern of length k in a text of length n in $O(k\log(n))$ time (using a simple binary search algorithm). The slower search time (compared to querying a suffix tree) is compensated by a three- to five-fold decrease in memory, requiring only 4 bytes per nucleotide in its basic form. An indirect approach for constructing a suffix array is to simply traverse a suffix tree using preorder traversal and record the values stored at each leaf node. For example, a preorder traversal of the tree in Figure 1.12 will yield the suffix array $\{9,8,4,6,2,5,7,3,1\}$ shown in Figure 1.13. However, more direct and lightweight methods to contruct the suffix array have been introduced since [Puglisi et al., 2007].

1.4.3.3 Enhanced suffix array

In [Abouelhoda et al., 2004] it was shown that every suffix tree algorithm can be replaced with an equivalent algorithm based on an *enhanced suffix array*, a data structure consisting of a suffix array and additional tables which help to navigate it. The ESA was primarily invented to be a space efficient alternative to the suffix tree, although this depends on the memory requirements of additional tables. Using an enhanced suffix array, the time to search a pattern of length k in a text of length n can be reduced back down to $O(k+z)$, equivalent to a suffix tree. One of the routinely used auxiliary tables is the *Longest Common Prefix array* which stores the lengths of the longest common prefices between successive suffices of a suffix array, or $\text{LCP}[i] = \text{lcp}(x[\text{SA}[i\text{-}1]..n], x[\text{SA}[i]..\text{n}])$ for $1 < i \leq n$. For example, the LCP for position $i = 5$ in the string $x =$ tatagata\$ in Figure 1.13 is computed between the suffices 'ata\$' and 'atagata\$', such that $\text{LCP}[5] = \text{lcp}(\underline{\text{ata}}\$,\underline{\text{ata}}\text{gata}\$) = 3$. Both tools Vmatch [Abouelhoda et al., 2004] and Segemehl [Hoffmann et al., 2009] use enhanced suffix arrays rather than suffix trees to align short sequences, where Vmatch reports using ∼7 bytes per nucleotide and Segemehl ∼13 bytes per nucleotide. The difference in the memory requirements is due to the fact that Segemehl supports non-exact read mapping using a conceptual suffix-interval tree and thus their enhanced suffix array data structure is slightly more complex. In the same (*2004*) paper, it was shown that the enhanced suffix array can be combined together with another data structure called the *Burrows-Wheeler Transform* (BWT) [Burrows and Wheeler, 1994] to efficiently locate repetitive regions in a sequence. The BWT is a reversible rearrangement of characters in a string with the property of grouping together identical characters to facilitate more efficient text compression, such as done in the tool bzip2. Figure 1.14 shows how to obtain the BWT of the string $x =$ tatagata\$. In the following subsection we will describe the latest FM-index data structure which strategically combines the BWT and the suffix array to provide fast query searches with the added benefit of a compressed index.

1.4.3.4 FM-index

The FM-index is a compressed suffix array-like data structure which is based on the Burrows-Wheeler Transform and stems roots from traditional suffix trees and enhanced suffix arrays. In [Ferragina and Manzini, 2000], it was shown that the suffix array implicitly resides within the BWT and a new data structure called the FM-index can be used to perform fast searches on a compressed index. The equivalence property between the suffix array and BWT can be intu-

Figure 1.13: Illustration of the enhanced suffix array (ESA) for the sequence x = tatagata\$. The ESA consists of a suffix array (SA) and additional tables such as the longest-common-prefix (LCP) or Burrows-Wheeler-Transform (BWT). The SA is a list of positions of all suffices of x in lexicographical order. A preorder traversal of the suffix tree in Figure 1.12 will yield the same list as given in the SA[i] column. The LCP array stores the length of the longest common prefix between a pair of consecutive suffices in the suffix array. For example, LCP[5] = 3 because x[SA[5]..9] = $\underline{\text{ata}}$gata\$ and x[SA[4]..9] = $\underline{\text{ata}}$\$. The construction of the BWT array is given in Figure 1.14. The BWT is frequently used to quicken navigation of the suffix array.

i	SA[i]	LCP[i]	BWT[i]	x[SA[i]..9]
1	9	0	a	\$
2	8	0	t	a\$
3	4	1	t	agata\$
4	6	1	g	ata\$
5	2	3	t	atagata\$
6	5	0	a	gata\$
7	7	0	a	ta\$
8	3	2	a	tagata\$
9	1	2		tatagata\$

itively seen by looking at Step 2 of Figure 1.14, where the permuted strings of x = tatagata\$ contain within them all suffices of the suffix array in x[SA[i]..9] of Figure 1.13. By using two auxiliary tables supporting last-to-first BWT column mapping (see [Ferragina and Manzini, 2000]) along with the BWT, a binary search algorithm can be performed to quickly find all locations of a pattern of length k in the index of length n in $O(k + z \log^{\epsilon} n)$ time (where z is the number of occurrences as before and $\epsilon > 0$ is chosen at the time the FM-index is built). In most implementations, the FM-index requires only 0.5-2 bytes per nucleotide [Li and Homer, 2010] which makes it the smallest indexing data structure currently employed for performing exact pattern matching (and approximate with performance compromises, see Section 1.4.1) in biological sequence analysis. Almost all recent HTS alignment tools use the FM-index including Bowtie [Langmead et al., 2009], BWA-SW [Li and Durbin, 2009], SOAP2 [Li et al., 2009], Bowtie2 and GEM [Marco-Sola et al., 2012]

Figure 1.14: Construction of the Burrows-Wheeler Transform for the sequence x = tatagata\$. In bioinformatics, the BWT is often simultaneously used for text compression and indexing. To construct the BWT of x, first all rotations of x are determined (Step 1). Next, the rotations are sorted lexicographically (Step 2) and the last character of each rotation is taken to construct the BWT (Step 3). The resulting string BWT = attgtaa\$ groups together runs of similar characters which can be easier compressed using run-length encoding. Morever, the BWT can be reversed into the original string x using a simple inverse transformation algorithm (time to compute same as sorting in Step 2).

Step 1: generate all rotations	Step 2: sort sequences lexicographically	Step 3: print last character of every sorted sequence
tatagata\$	\$tatagat**a**	
atagata\$t	a\$tataga**t**	
tagata\$ta	agata\$ta**t**	
agata\$tat	ata\$tata**g**	
gata\$tata	atagata\$**t**	
ata\$tatag	gata\$tat**a**	**attgtaaa\$**
ta\$tataga	ta\$tatag**a**	
a\$tatagat	tagata\$t**a**	
\$tatagata	tatagata**\$**	

Chapter 2

New approximate seeding technique and supporting data structures

Contents

In this chapter we will describe a novel approximate seeding technique that can detect whether two strings W and V have an edit distance $d(W,V) \leq 1$. The *edit distance* $d(W,V)$ measures the amount of difference between two strings. It is defined as the minimum number of edit operations needed to transform one string into the other – with allowable edit operations being *insertion*, *deletion* or *substitution* of a single character. Figure 2.1 shows various examples of strings W and V with edit distance $d(W,V) = 1$.

```
W = A-CCTGA       CTAGGATAA       GACACATT
    | |||||       ||||*||||       ||||| ||
V = ATCCTGA       CTAGCATAA       GACAC-TT
```

Figure 2.1: Example of edit distance $d = 1$ for various strings.

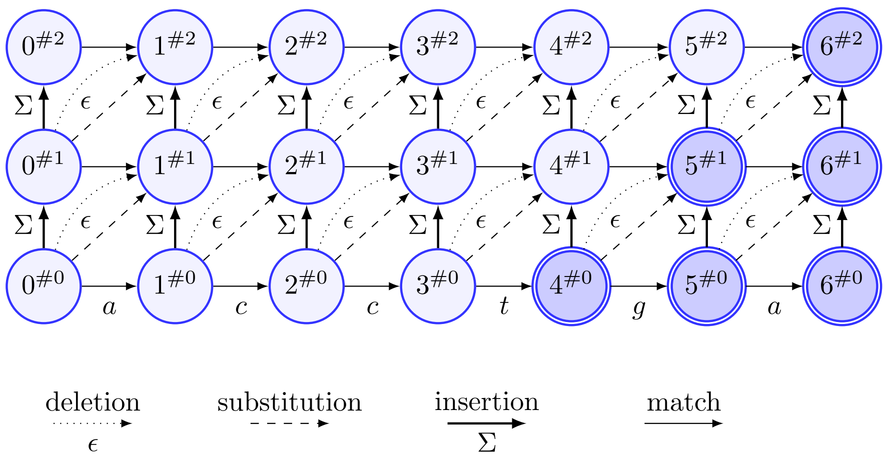
We apply this seeding technique in *approximate pattern matching* for querying a string in a large text allowing up to 1 error of any type. Given a query W and a reference text R, we want to find all occurrences of $R[i,j]$ such that $d(W,R[i,j]) \leq 1$.

In Section 2.1 we describe the technicalities of our novel seeding method using the universal Levenshtein automaton for $d = 1$. In Section 2.2.1 we go on to describe the indexing data structures used together with the universal Levenshtein automaton to find k-mers allowing up to 1 error in a reference text.

2.1 The Levenshtein automaton

The classical nondeterministic Levenshtein automaton for a pattern p and a number of errors d recognizes the set of strings which are at most edit distance d to p (see Figure 2.2). This automaton is not suitable for large-scale computations because of the presence of multiple active states (enhanced by epsilon transitions) which represent all feasible scenarios of alignments between two strings. Epsilon transitions allow for states to be activated without consuming any input character. In the Levenshtein automaton their purpose is to consider possible deletions of characters in both strings. For example, prior to searching the query $q = ctga$ in the automaton for the pattern $p = acctga$ of Figure 2.2, the states $0^{\#0}$,$1^{\#1}$ and $2^{\#2}$ will be activated before the first letter $q[0] = c$ is input into the automaton. In this manner, active state $2^{\#2}$ considers two possible deletions in q prior to evaluating a state transition for input letter c (which would result in a match from $2^{\#2} \rightarrow 3^{\#2}$).

Figure 2.2: The nondeterministic Levenshtein automaton for $p = acctga$ and $d = 2$. The $s^{\#e}$ notation for each state corresponds to s number of characters read in the pattern p and e number of errors recorded. The initial state is $0^{\#0}$ and the six final states are $4^{\#0}$, $5^{\#0}$, $5^{\#1}$, $6^{\#0}$, $6^{\#1}$ and $6^{\#2}$. Each non-final state has three outgoing arcs, one for each type of edit operation.



A common solution to deal with the multiple active states is to transform the nondeterministic automaton into an equivalent deterministic one using classical powerset construction [Hopcroft et al., 2004], which will contruct a new deterministic state for every possible set of nondeterministic active states. However, the resulting automaton may be exponential in the length of p and it will continue to be limited to representing strings of that defined length. In [Schulz and Mihov, 2002] and [Mihov and Schulz, 2004] a universal Levenshtein automaton was introduced based upon insightful observations of the classical one. The term *universal* conveys its one-time construction and independency of p, thus it can be applied to any pair of strings of arbitrary lengths. The intuition arises from the symmetry of the nondeterministic automaton, which applies the same set of transition rules to every new input character and each new set of active states is a subset of a known bounded superset. Formally, these bounded supersets are called *symbolic triangular areas* and defined in Definition 1.

Definition 1. [Mihov and Schulz, 2004] Let p denote a pattern of length ρ and $A(p, d)$ the nondeterministic Levenshtein automaton for error d. The **triangular area of a state** λ of $A(p, d)$ consists of all states β of $A(p, d)$ that can be reached from λ using a (potentially empty) sequence of u upward transitions and, in addition, $h \leq u$ horizontal or reverse (i.e., leftward) horizontal transitions. Let $0 \leq i \leq \rho$. By **triangular area** i, we mean the triangular area of state i^0. For $j = 1, \ldots, d$, by **triangular area** $\rho + j$, we mean the triangular area of state ρ^j.

To highlight the idea behind this principle, let us search any query in the generic nondeterministic automaton for $\rho = 7$ and $d = 2$ illustrated in Figure 2.3 (meaning the same Levenshtein automaton for $d = 2$ can be applied to any pattern p of length $\rho = 7$). After reading the 2^{nd} character in our query, the set of all possible active states is bounded by the yellow symbolic triangular area (A). By reading the 3^{rd} character, the symbolic triangular area will shift one position to the right. The semantics of the Levenshtein automaton transitions enforce that the horizontal span of all reachable states is $\leq 2d + 1$. For example for $d = 2$, the base of the triangular area will span 5 states (see Figure 2.3). The new set of active states for each triangular area only depends on the previous set of active states and the new transitions taken by reading any character. As long as the number of characters read is $< \rho - d$, the new subset of active states will always contain *non-accepting* (I states). Beginning at the $(\rho - d)^{th}$ character, the triangular area will also contain *accepting* states (M states), see Figure 2.4. Together, these general observations have led to the formulation of a universal Levenshtein automaton. In full generality, the size of the automaton is exponential in a function of d. More specifically, the total number of deterministic states is classified by $O((d+1)2^{4d - \log_2 \sqrt{2d+1}})$ [Mitankin, 2005]. Therefore, the automaton grows quickly for higher number of errors, where for $d = 1$ there are only 14 states and for $d = 2$ or $d = 3$ there are already 90 or 602 states respectively.

A set of characteristic vectors symbolizing the homology of p and a word W serve as the alphabet to the automaton and must be precomputed using Definition 2.

Definition 2. [Mihov and Schulz, 2004] The **characteristic vector** $\vec{\chi}(w, V)$ of a symbol $w \in \sum$ in a word $V = v_1 \ldots v_n \in \sum^*$ is the bitvector of length n where the ith bit is set to 1 iff $w = v_i$.

The length of the bitvectors is $\leq (2d + 1) + 1$, where $2d + 1$ is the longest span of reachable states in the nondeterministic Levenshtein automaton and the additional sum of 1 is for the last bit to identify the transition between non-accepting and accepting states.

Let $d = 1$, the input word $W = acaga$ and the pattern $p = \$actaga$ (d number of \$ characters are added to the prefix to standardize the length of bitvectors obtained for the initial characters), then $\chi_1(\underline{a}, \$\underline{a}ct) = 0\underline{1}00$, $\chi_2(\underline{c}, a\underline{c}ta) = 0\underline{1}00$, $\chi_3(\underline{a}, ct\underline{a}g) = 00\underline{1}0$, $\chi_4(\underline{g}, ta\underline{g}a) = 00\underline{1}0$, $\chi_5(\underline{a}, \underline{a}g\underline{a}) = \underline{1}0\underline{1}$ are the computed characteristic bitvectors. It follows that $\{\chi_1, \ldots, \chi_5\}$ is the *characteristic bitvector array* carrying the similarity information of W and p.

Beginning from χ_1 to $\chi_{|W|}$, the bitvectors are sequentially passed into the universal Levenshtein automaton and serve as the new alphabet (rather than the letters themselves). Each bitvector leads to a transition between states in constant time (provided that the bitvector fits into a computer word) corresponding to the number of errors encountered thus far. If some χ_i reaches a failure state, greater than d errors exist between W and p, and the strings are rejected. The automaton only recognizes two strings if the input of the last bitvector $\chi_{|W|}$ leads to a final state. An example of bitvector computations and automaton traversal for the two words $p = acctga$ and $W = atcctga$ is shown in Figure 2.5.

The mathematics behind constructing this automaton are well described in [Schulz and Mihov, 2002, Mihov and Schulz, 2004], however here we also give a simple example of how the universal property can be identified using a classical non-deterministic Levenshtein automaton. Figure 2.6 illustrates a sequence of snapshots for the active states of a non-deterministic Levenshtein automaton for $d = 1$ for the word $W = acgt$ and the input pattern $p = agt$ as p is introduced into the automaton character by character. The universal Levenshtein automaton determinizes the active states in the yellow triangle, and the changes of transitions are carried out using characteristic bitvectors (in red) rather than individual letters of the input pattern.

2.1.1 Application in biological sequence alignment

The ability of the universal Levenshtein automaton to quickly recognize two strings with edit distance $\leq d$ is an extremely valuable property for developing new approximate seeds allowing indel errors. We chose to work with $d = 1$ as the corresponding automaton is very small and the search time is up to 20x faster than for $d = 2$ and 60x faster than for $d = 3$ [Mihov and Schulz, 2004]. However, higher error automata may be useful for more specialized applications such as tracking dinucleotide mutations or identifying variations in protein coding regions. Our approach to using a universal seed with $d = 1$ is to search for short k-mer matches for $k \in [8, 26]$ between a read and a database of reference sequences allowing up to 1 error. Since the input to the universal Levenshtein automaton is a series of characteristic bitvectors computed from two strings (rather than the strings themselves), we had to determine a clever technique to quickly build these bitvectors between any k-mer on the read and those found in the reference database. For this, we devised the *dynamic bitvector table* which is illustrated in Figure 2.7.

Going back to the problem of finding all *approximate* k-mer occurrences in a text (see Section 1.4.1), we can precompute a bitvector table of size $4k$ (for DNA text) and use it to retrieve any combination of characteristic bitvectors for all 4^{k+1} possible matches (a match is defined to have ≤ 1 error to the k-mer) in the text *without* actually ever seeing the text. During the search for a k-mer in a text, we can quickly retrieve the correct bitvector corresponding to a character being observed in the text without needing to recompute the bitvector for every new candidate. Initially, the dynamic bitvector table is computed for the first k-mer on the read where this k-mer is equivalent to the word V in Definition 2 and each input symbol $w \in \{a, c, g, t\}$ belongs to a k-mer in the reference text (see Figure 2.7). Thusly, in order to search for all k-mers on the read in the reference text, the bitvector table can be shifted by one position to the right on the read and updated using bitwise operations to remove the first character of the preceding k-mer and add a new character to the suffix. This operation can be seen in Figure 2.8.

By continuously shifting the bitvector table along the read, a new characteristic bitvector table can be quickly precomputed for each k-mer using the previous one and used to search the reference text. As discussed in Section 1.4, naïvely searching for a query k-mer in a large text by comparing each k-mer against all possible candidates is extremely inefficient, thus we have also developed a new lossless text indexing data structure for rapidly listing all k-mer matches. This reference index is constructed one time and may be reused for any set of reads. The following subsections describe this indexing data structure and its main utilization in searching for seeds.

2.2 Indexing with the Burst trie

In Section 1.4.3 we have seen that tree-like index data structures, such as the suffix tree, are suitable for approximate seeds. However, suffix trees require large memory resources and more suitable data structures exist for searching large collections of *identical length* strings. Here, we propose to use an alternative data structure, namely the Burst trie [Heinz et al., 2002]. Futher-

Figure 2.3: Symbolic triangular area for non-accepting states of type I. The yellow symbolic triangle (A) represents the set of all *possible* active states after the **2nd** letter x_2 of an input word was read by the automaton. Similarly, after reading the **3rd** letter x_3, the triangular area shifts one position to the right. The new triangular area (B) will encompass all new states reached by transitions from triangular area (A). Since neither (A) nor (B) contain a final state of type M, they fall under the same name of "symbolic triangular area for non-accepting states of type I".

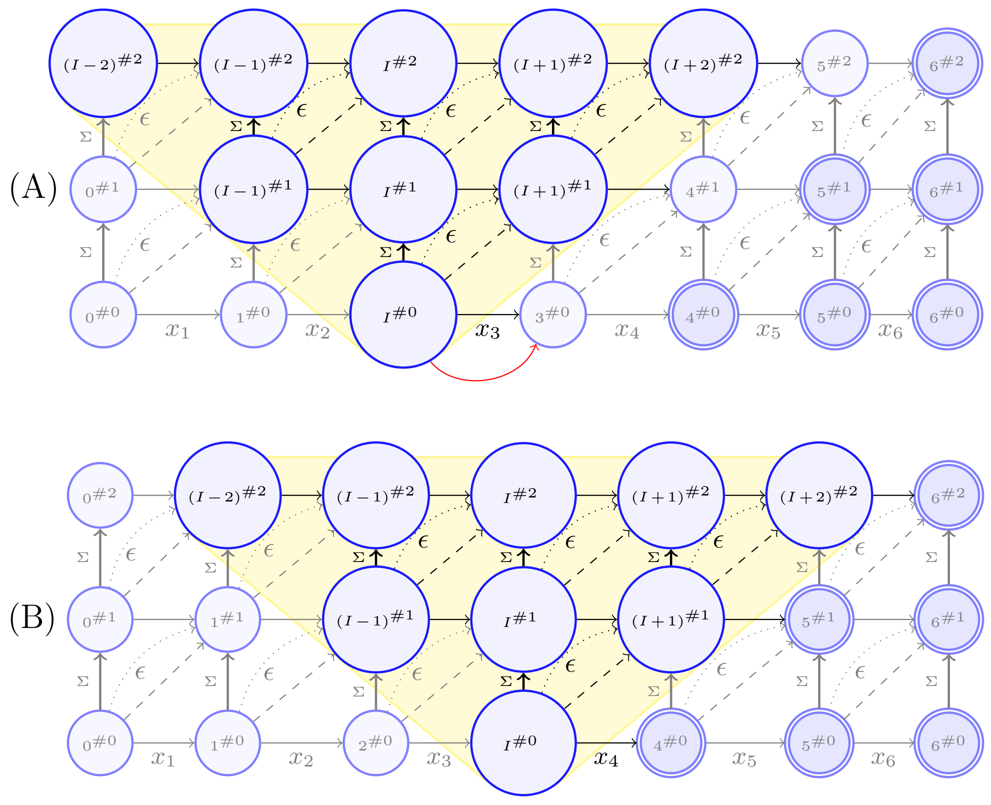

(A)
$(I-2)^{\#2}$ $(I-1)^{\#2}$ $I^{\#2}$ $(I+1)^{\#2}$ $(I+2)^{\#2}$ $5^{\#2}$ $6^{\#2}$
$0^{\#1}$ $(I-1)^{\#1}$ $I^{\#1}$ $(I+1)^{\#1}$ $4^{\#1}$ $5^{\#1}$ $6^{\#1}$
$0^{\#0}$ $1^{\#0}$ $I^{\#0}$ $3^{\#0}$ $4^{\#0}$ $5^{\#0}$ $6^{\#0}$
x_1 x_2 x_3 x_4 x_5 x_6
Σ ϵ
(B)
$0^{\#2}$ $(I-2)^{\#2}$ $(I-1)^{\#2}$ $I^{\#2}$ $(I+1)^{\#2}$ $(I+2)^{\#2}$ $6^{\#2}$
$0^{\#1}$ $1^{\#1}$ $(I-1)^{\#1}$ $I^{\#1}$ $(I+1)^{\#1}$ $5^{\#1}$ $6^{\#1}$
$0^{\#0}$ $1^{\#0}$ $2^{\#0}$ $I^{\#0}$ $4^{\#0}$ $5^{\#0}$ $6^{\#0}$
x_1 x_2 x_3 x_4 x_5 x_6
Σ ϵ

Figure 2.4: Symbolic triangular area for accepting states of type M. The yellow symbolic triangle represents all *possible* active states after the **4th** letter x_4 was read by the automaton. This triangular area encompasses three of the six final non-deterministic states $\{(M-2)^{\#0}, (M-1)^{\#1}$ and $M^{\#2}\}$ and is known as the *accepting-state* triangular area. All deterministic states derived from this area will form *accepting states* in the universal Levenshtein automaton for $d = 2$.

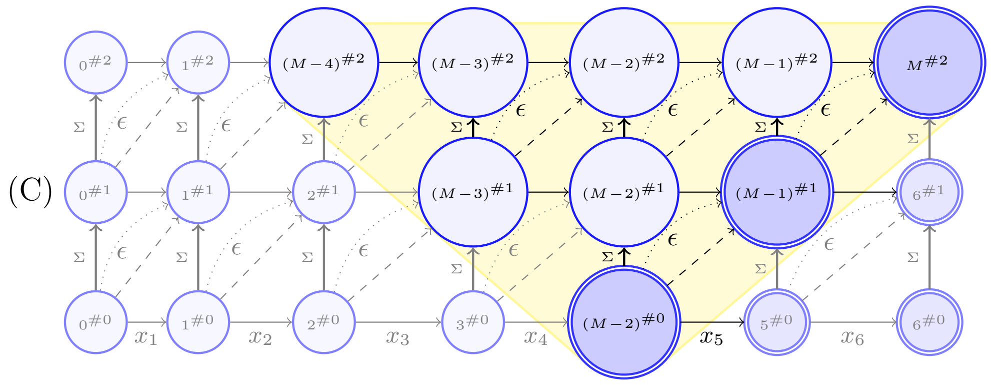

(C)
$0^{\#2}$ $1^{\#2}$ $(M-4)^{\#2}$ $(M-3)^{\#2}$ $(M-2)^{\#2}$ $(M-1)^{\#2}$ $M^{\#2}$
$0^{\#1}$ $1^{\#1}$ $2^{\#1}$ $(M-3)^{\#1}$ $(M-2)^{\#1}$ $(M-1)^{\#1}$ $6^{\#1}$
$0^{\#0}$ $1^{\#0}$ $2^{\#0}$ $3^{\#0}$ $(M-2)^{\#0}$ $5^{\#0}$ $6^{\#0}$
x_1 x_2 x_3 x_4 x_5 x_6
Σ ϵ

Figure 2.5: Conceptual example of the universal deterministic Levenshtein automaton for $d = 1$ (not all of the transitions are shown). To see whether the word W = atcctga and the pattern p = acctga have Levenshtein distance ≤ 1, we first compute the set of characteristic bitvectors representing them using Definition 2. The resulting characteristic bitvector array is $\chi = \{0100, 0001, 1100, 1000, 100, 10, 1\}$. In the universal Levenshtein automaton, the I states are the deterministic non-accepting states of the symbolic triangular area shown in Figure 2.3. Similarly, the M states are the deterministic accepting states of the symbolic triangular area shown in Figure 2.4. The transitions between states are made using one bitvector from the characteristic bitvector array. For each transition, the x character in the bitvector is a joker and will accept both a '0' or a '1' in its corresponding position. Moreover, if the joker character is found inside brackets, i.e. (x), then it is not obligatory to exist in the bitvector. For example, the transition labeled as $x1x(x)$ will accept the bitvectors 0101, 1100 and 111 (amongst other possibilities). However, it will not accept the bitvectors 1001, 1000 and 11 (amongst other possibilities). Beginning from the initial state (labeled 'start'), each bitvector in χ is input to the automaton in order and the transitions are followed accordingly. If the last bitvector transition leads to an M state, then W and p match with $d \leq 1$. Otherwise if a null state is reached (meaning there does not exist a transition from the current state corresponding the next bitvector) or an I state is reached using the last bitvector in χ, then W and p do not match with $d \leq 1$.

Step	Precomputed bitvectors		State transitions
(a)	$\chi_1(a, \$acc)$	$= 0100$	$\{I^{\#0}\} \xrightarrow{0100} \{I^{\#0}\}$
(b)	$\chi_2(t, acct)$	$= 0001$	$\{I^{\#0}\} \xrightarrow{0001} \{(I-1)^{\#1}, I^{\#1}\}$
(c)	$\chi_3(c, cctg)$	$= 1100$	$\{(I-1)^{\#1}, I^{\#1}\} \xrightarrow{1100} \{(I-1)^{\#1}, I^{\#1}\}$
(d)	$\chi_4(c, ctga)$	$= 1000$	$\{(I-1)^{\#1}, I^{\#1}\} \xrightarrow{1000} \{(I-1)^{\#1}\}$
(e)	$\chi_5(t, tga)$	$= 100$	$\{(I-1)^{\#1}\} \xrightarrow{100} \{(I-1)^{\#1}\}$
(f)	$\chi_6(g, ga)$	$= 10$	$\{(I-1)^{\#1}\} \xrightarrow{10} \{(I-1)^{\#1}\}$
(g)	$\chi_8(a, a)$	$= 1$	$\{(I-1)^{\#1}\} \xrightarrow{1} \{M^{\#1}\}$

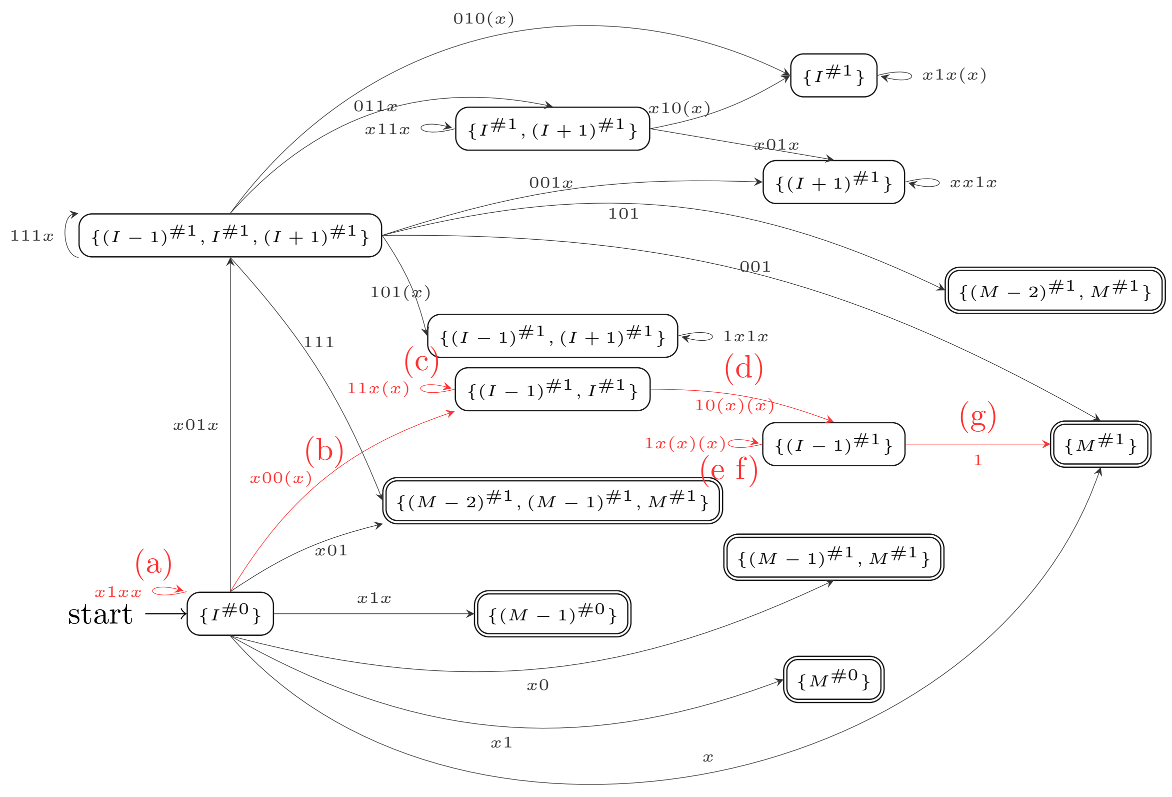


Figure 2.6: A non-deterministic Levenshtein automaton for the word $w = acgt$. The $s^{\#e}$ notation for each state corresponds to s number of characters read in the pattern p and e number of errors recorded. The initial state is $0^{\#0}$, the final states are $3^{\#0}$, $4^{\#0}$ and $4^{\#1}$, and the active states are illustrated in blue color. The yellow triangle represents the boundary of all possible active states after a character is consumed by the automaton. The pattern p to be consumed is agt. The red binary sequences are the characteristic bitvectors between the input pattern agt and the automaton word $acgt$, defined in Section 2.3 of the paper. If a bit of a bitvector is set to 1, the match transition is possible for active states in the adjacent left column of the bit in the automaton. Otherwise, if a bit is set to 0, the match transition is not permitted. Each step corresponds to consuming one character of agt by the automaton.

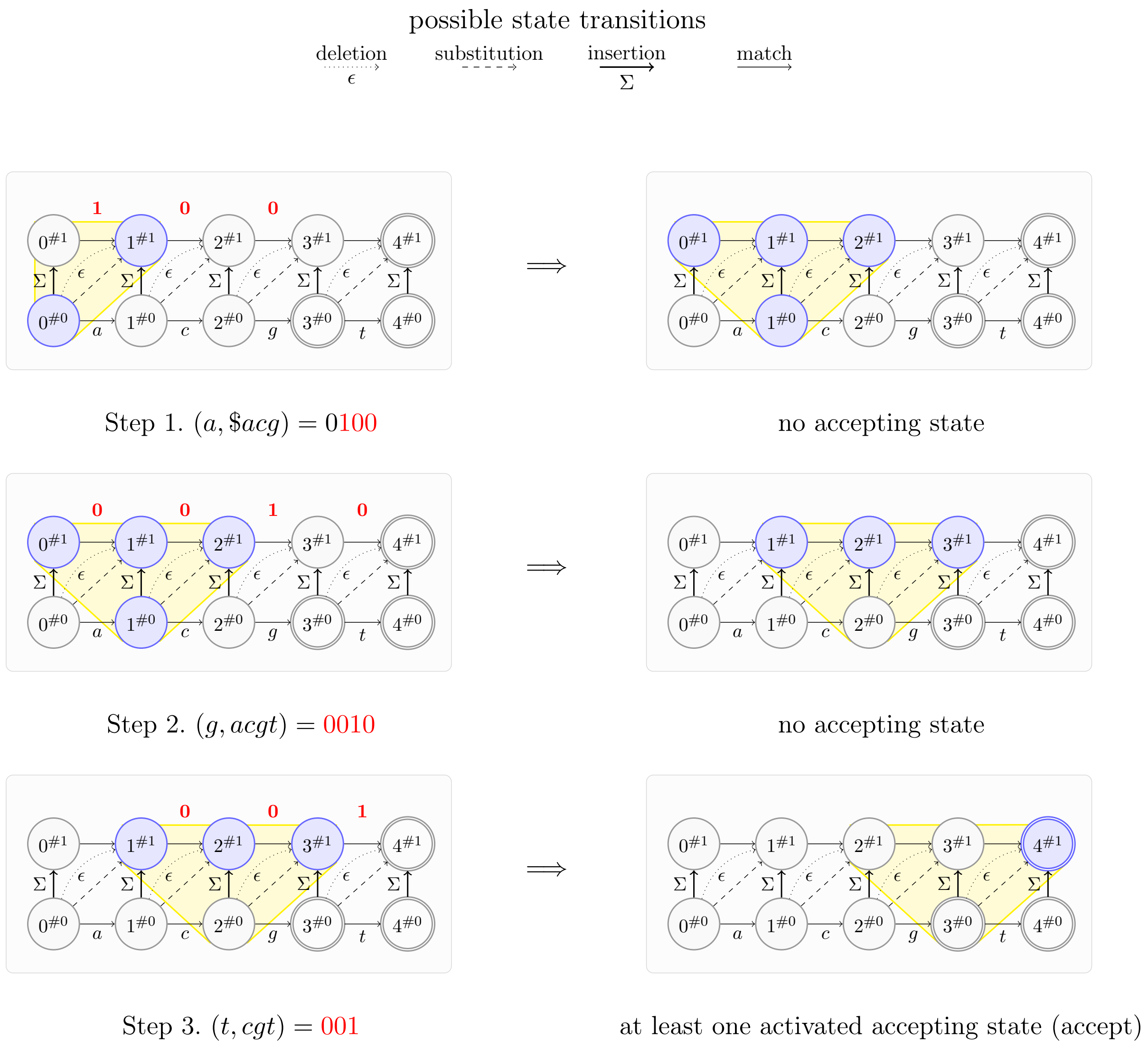

possible state transitions
deletion ϵ
substitution
insertion Σ
match
Step 1. $(a, \$acg) = 0100$
no accepting state
Step 2. $(g, acgt) = 0010$
no accepting state
Step 3. $(t, cgt) = 001$
at least one activated accepting state (accept)

Figure 2.7: The precomputed bitvector table for pattern $p = \$actaga$ covering all possibilities of q for $d = 1$. The first bit in each entry of column $i = 0$ represents the \$ symbol and is always set to '**0**'. If the query $q = actag$ was being searched, then the highlighted set of bitvectors $0100, 0100, 0010, 0010, 101$ would be passed to the universal Levenshtein automaton.

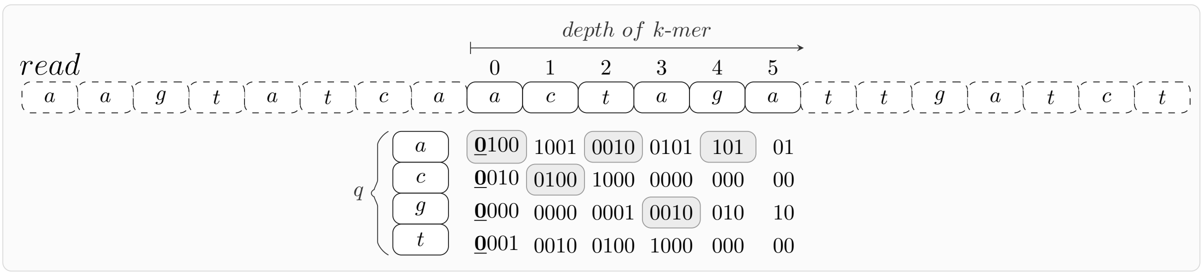
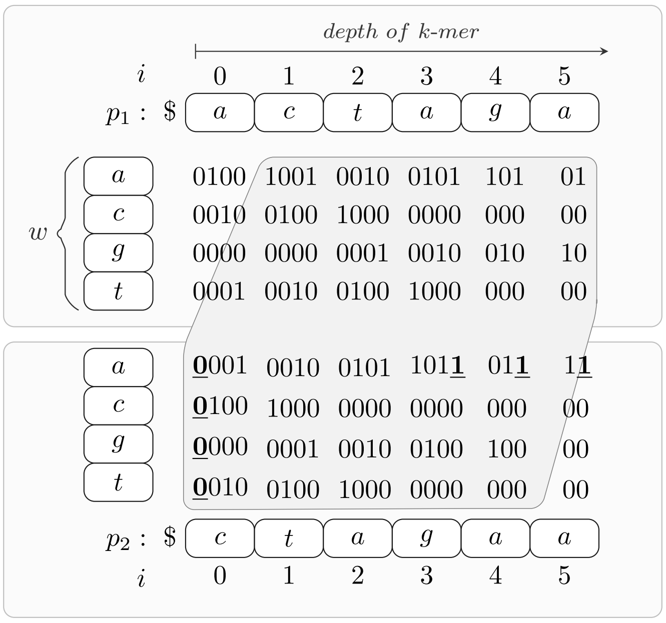


Figure 2.8: The modification of the bitvector table from pattern $p_1 = \$actaga$ to $p_2 = \$ctagaa$ for $d = 1$. Columns 0-2 of p_2 are equal to columns 1-3 of p_1, except for column 0, where the most significant bit (MSB) of every bitvector represents the symbol \$ and is set to '**0**'. Column 3 of p_2 equals to column 4 of p_1 with an additional bit appended. The appended bit is set to '**1**' in the bitvector corresponding to the newly appended character, otherwise it is set to 0. Column 4 of p_2 is equal to column 3 of p_2, although the MSB is not considered. The same rule applies to column 5 of p_2, where the two MSBs of the column 3 bitvectors are not considered.



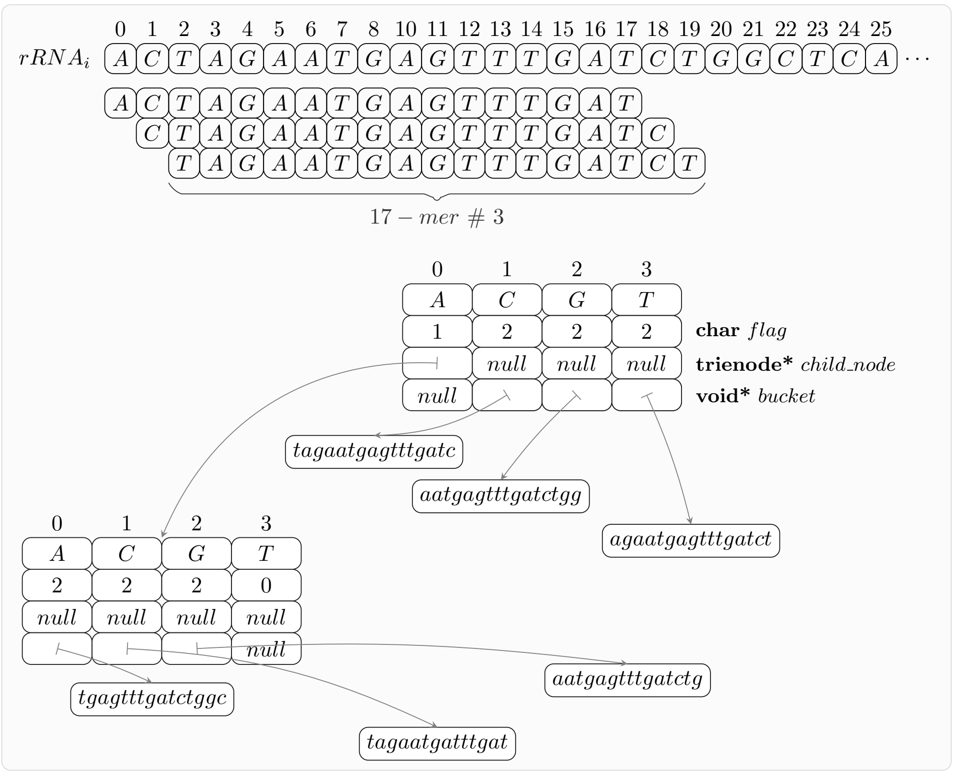
more, we explain how to optimize this data structure for searching words of length k allowing up to one error.

2.2.1 The Burst trie

The Burst trie is a fast and versatile data structure which effectively stores large numbers of strings such as an rRNA database or large collections of genomic sequences. Given a sequence $x = ab$, the Burst trie can store the prefix a as a link of trie nodes and the suffix b as an array of characters appended to the last trie node: this extension to the last node is called a "bucket" (see

Figure 2.9). Normally, subtrees become more sparse in the depth of a trie and representing them as reduced "buckets" of contiguous memory preserves space and boosts cache-efficiency. When the number of sequences sharing a common prefix a reaches a fixed threshold, the appended bucket of suffixes bursts to form a new trie node and smaller sub-buckets. To optimize memory access during subtree traversal, the threshold size of a bucket should be less than the lower level cache. A systematic use of this trie can be observed in the fastest sorting algorithm for large sets of strings, the Burstsort [Sinha and Zobel, 2004].

Figure 2.9: Let $k = 16$, the Burst trie below is constructed on the first six 17-*mers* of a reference sequence. The '**char** *flag*' describes whether a pointer is set to a trie node '1', a bucket '2' or neither '0'. Additional information on the origin of the 17-mer directly follows each element, as shown in the dashed bucket.



2.2.2 Improvement: Lookup table & mini-burst tries

We use an additional optimization to improve access into the Burst trie. Since we consider at most one error between the window and the database, we have this simple property: For every two words of length $\sim k$ such that the edit distance between them is bounded by 1, there exists a common substring of length $\frac{k}{2}$ which is either a prefix or a suffix of the two words. We apply this property to construct a lookup table storing all $\frac{k}{2}$-mers existing in the reference database. Note that for $k \in [8, 26]$, transposing the nucleotide alphabet onto a binary equivalent, such that $\{a, c, g, t\} = \{00, 01, 10, 11\}$, we can represent each $\frac{k}{2}$-mer in k bits which maps to a unique integer value. Upon completion of the forward and reverse Burst tries, a scan of each trie is performed to record the existence of all $\frac{k}{2}$-mers and, if present, associated pointers to the trie node representing the immediate letter following the prefix. The precomputed lookup table quickly determines whether an exact match of the prefix or suffix exists in the Burst tries and furthermore it provides us with direct access to the remaining part of the word in the Burst trie.

2.2.3 Implementation

Following a similar method of an array-structured trie as described in [Sinha et al., 2006], our Burst trie is assembled exactly on the nucleotide alphabet $\{a, c, g, t\}$. As illustrated in Figure 2.9, the trie stores every unique $(k+1)$-mer substring in a reference database, since we look at seeds of length k with at most 1 error between any two words (note that the extra nucleotide for a reference $(k+1)$-mer is to account for a possible insertion or deletion of a nucleotide in the query k-mer). The information on whether the $(k+1)$-mer belongs to a forward strand, the reverse-complement or both (*strand*), and its origin (*hashid*) follows each entry in a bucket (not shown in Figure 2.9). When the exact location of the $(k+1)$-mer needs to be found in the reference database, the *hashid* value serves as an index in a complementary table storing this information. For databases containing highly conserved sequences, such as the 16 rRNA for which nearly one-quarter of the positions are $99-100\%$ conserved [Cannone et al., 2002, Mears et al., 2002], this data structure moderates the size of the tree since many identical or closely similar substrings are shared between sequences.

In our initial implementation of the index we constructed one Burst trie on all possible unique $(k+1)$-mers in the reference database and used a $\frac{k}{2}$-mer lookup table to quickly access different nodes of the Burst trie. However, this approach can be further improved by "cutting off" the trie nodes from the root of the Burst trie to depth k and replacing them by a lookup table. Then, every entry in the lookup table is connected to a separate (mini) burst trie, which represent the children nodes (below depth k) of the original full trie. In the following subsections we will describe how such a data structure can be easily computed directly, without the prior need to explicitly construct a full Burst trie, to both reduce the memory requirements and speed up the search by optimizing calls to the local cache.

Index construction for the reference database

To find all occurrences of a k-mer in a reference database with an edit distance of at most 1, we apply the pigeonhole principle to divide the k-mer into two equal parts, such that at least one part will match with 0 error. This property allows us to take advantage of the following arborescent data structure setup:

1. build a $(\frac{k}{2})$-mer exact match lookup table (see Figure 2.10(2a))

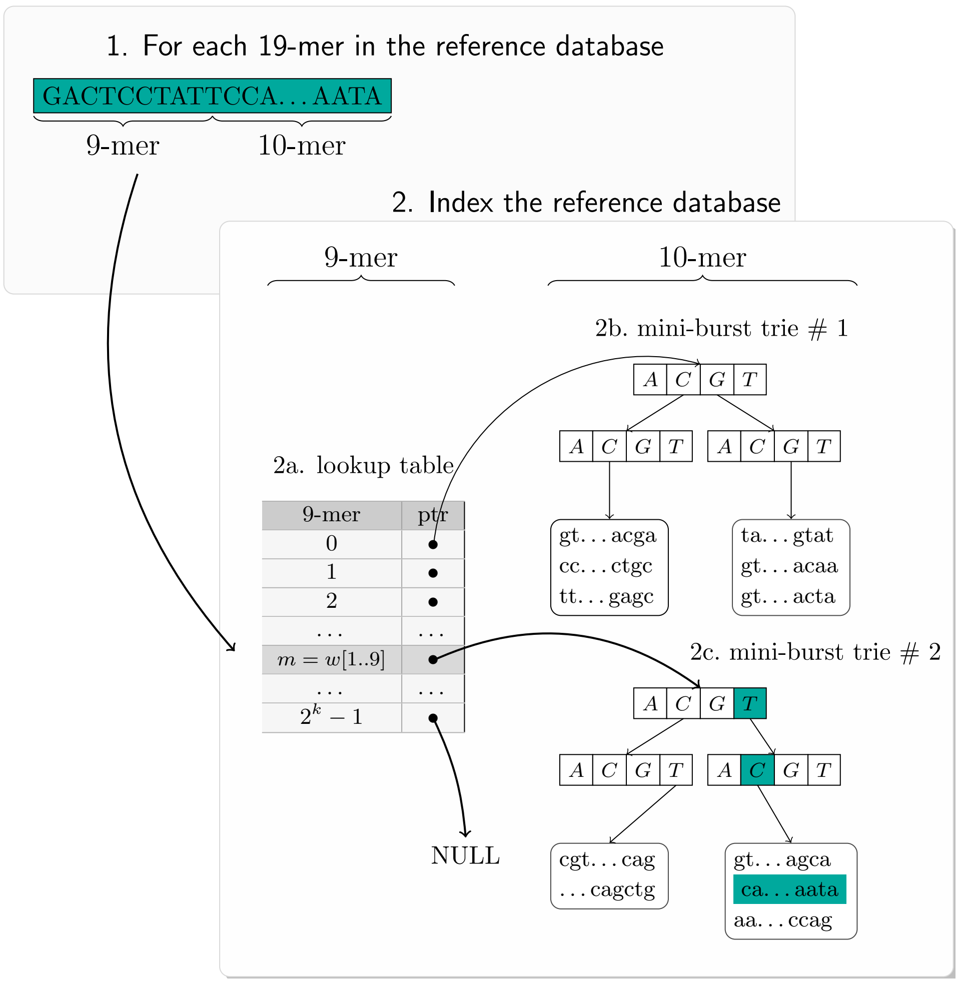
2. connect each entry in the lookup table to a *mini-burst trie* storing the remaining $\frac{k}{2}+1$ characters and allowing search up to one error (see Figure 2.10(2b/c))

The arborescent index was inspired by the burst trie data structure, but with a twist to index the prefixes of strings with a lookup table and the suffixes in a collection of branching (mini) burst tries. The burst trie data structure was chosen due to its relatively low memory footprint and fast access to each entry.

The $(\frac{k}{2})$-mer lookup table and the $(\frac{k}{2}+1)$-mer mini-burst tries work hand-in-hand to store all unique $(k+1)$-mers of the reference database, and a separate list of positions gives occurrences for all unique $(k+1)$-mers. The $(\frac{k}{2})$-mer lookup table is accessed by converting a $(\frac{k}{2})$-mer into a unique decimal value using 2 bits per nucleotide encoding, where $A = 00$, $C = 01$, $G = 10$ and $T = 11$. For example the lookup index i for the string $x = ACTAGTATT$ would be $i = 000111001011001111 = 29391$.

This arborescent data structure was designed to optimally enumerate all $(k+1)$-mer sequences which match to a k-mer query sequence allowing up to 1 error, note that the extra nucleotide for a reference $(k+1)$-mer is to account for a possible insertion or deletion of a nucleotide in the query k-mer. The original positions of all $(k+1)$-mers in the reference index are stored in a separate inverted list of positions. The index for this list is computed using the CHM minimal perfect hash function (CMPH library [Reis et al., 2012]) and stored next to each $(\frac{k}{2}+1)$-mer entry in the mini-burst trie leaves. In future implementations, the size of this auxiliary positions list can possibly be reduced in size by using a pairing function [Lisi, 2007].

Figure 2.10: Lookup table and mini-burst trie index



Due to the dynamic nature of index construction (self-adjusting tree composed of trie nodes and bucket nodes), one cannot precisely calculate the size of the index before it is built. However, from multiple simulations on varying data, the estimated size has shown to be inferior to

100×(length of reference sequence) bytes. To keep the index of a practical size, it can be divided into multiple subparts. During the mapping of reads, each index subpart can be sequentially loaded and processed in memory, always maintaining the memory for the index below its designated threshold. Although the optimal mapping time is achieved if the index can be constructed in one part, the overhead time with processing multiple subparts is not impractical. Thus, the index fragmentation feature nonetheless allows convenient utilization of this data structure.

2.2.4 Searching for matching seeds in the reference index

The search for a k-mer seed in the indexed database is performed in two steps. Firstly, the k-mer is split into two equal parts and the first part is directly searched in the $(\frac{k}{2})$-mer lookup table (with 0 errors). Then, if the $(\frac{k}{2})$-mer exists, a pointer redirects the remaining of the search to a corresponding mini-burst trie (allowing up to 1 error). Here, a parallel traversal is performed between the second part of the k-mer against all entries in the mini-burst trie using the universal Levenshtein automaton for edit distance $d = 1$. At every depth of the mini-burst trie, we assume that the symbol q in $\chi_i(q, V)$ appears as one of $\{a, c, g, t\}$ with equal probability. Following a pre-order path, the traversal of the Burst trie begins at the root node. Through knowledge of the nucleotide letter and the depth of the node being visited, the coinciding bitvector is accessed in the precomputed bitvector table, indifferent to whether the node is a trie node or a character in the bucket. Subsequently, the bitvector is passed to the universal Levenshtein automaton which decides whether to continue traversal of the current subtree or backtrack to the first branching point with a non-failure Levenshtein state and recommence traversal of a new substree. In this manner, a complete traversal of the mini-burst trie remains unlikely as backtracking occurs each time the edit distance between the pattern and a traversed branch exceeds k. To further speed up Burst trie traversal for every k-mer, a 'backwards dictionary' approach as described in [Mihov and Schulz, 2004] was implemented. This means that we build mini-burst tries for the forward and reverse sense of $(k + 1)$-mer in the reference database so that it can be traversed quickly from both ends.

2.3 Extending seeds into longer matches using the LIS

The Smith-Waterman dynamic programming algorithm is the most precise method to score a homology between two sequences and is used in the majority of short-read alignment tools after the intial seeding step. Here we describe our method to use the collection of matching seeds accumulated during mini-burst trie traversal to isolate longer homologous regions between the read and a reference sequence prior to performing localized Smith-Waterman alignment. We begin by binning the seeds to their corresponding reference sequences by using the $(k + 1)$-mer list of positions to link each matching seed to the original location on a reference sequence as illustrated in Figure 2.11. Next, we isolate regions of length equal to the read on each reference sequence containing a threshold number of matching seeds (by default 2). The next step involves computing variant of the *longest increasing subsequence* (LIS) on the seeds' positions on the read relative to a region on the reference sequence, as shown in Steps 3-4 of Figure 2.11. The LIS is the longest subsequence of elements in a sequence where the elements in the subsequence are in increasing (sorted) order. For example for the given the sequence of integers $\{7, 1, 9, 6, 42, 23, 40, 26\}$ one possible LIS is $\{1, 6, 23, 40\}$. By first sorting the (position on reference - position on read) k-mer positions using the 'position on reference' as a key and then computing the LIS on the reads' positions, we try to reassemble our collection of k-mers into a longer contiguous region common to both sequences. Finally, if the LIS is composed of a threshold number of seeds (by default 2), we proceed to the final step of Smith-Waterman alignment beginning near the LIS.

2.4 Conclusion

In this chapter we introduced a novel approximate seeding technique along with the complementary machinery optimized for its efficiency. We described a new index data structure never used before in short-read sequence alignment which can accommodate seeds allowing up to 1 error of any type. Moreover, the error is not restricted to any predefined position in the seed which gives it flexibility for unpredictable error distribution. The tradeoff for maintaining such seeds is the size of the new index data structure which is able to accomodate searches with insertions and deletions, requiring more space than the BWT. However, due to the malleability of the data structures used, the index can be divided into multiple subparts without significant effects on its performance.

⋆ ⋆ ⋆

Figure 2.11: How to extend multiple k-mer matches into a longer homologous region using the longest increasing subsequence of k-mer positions

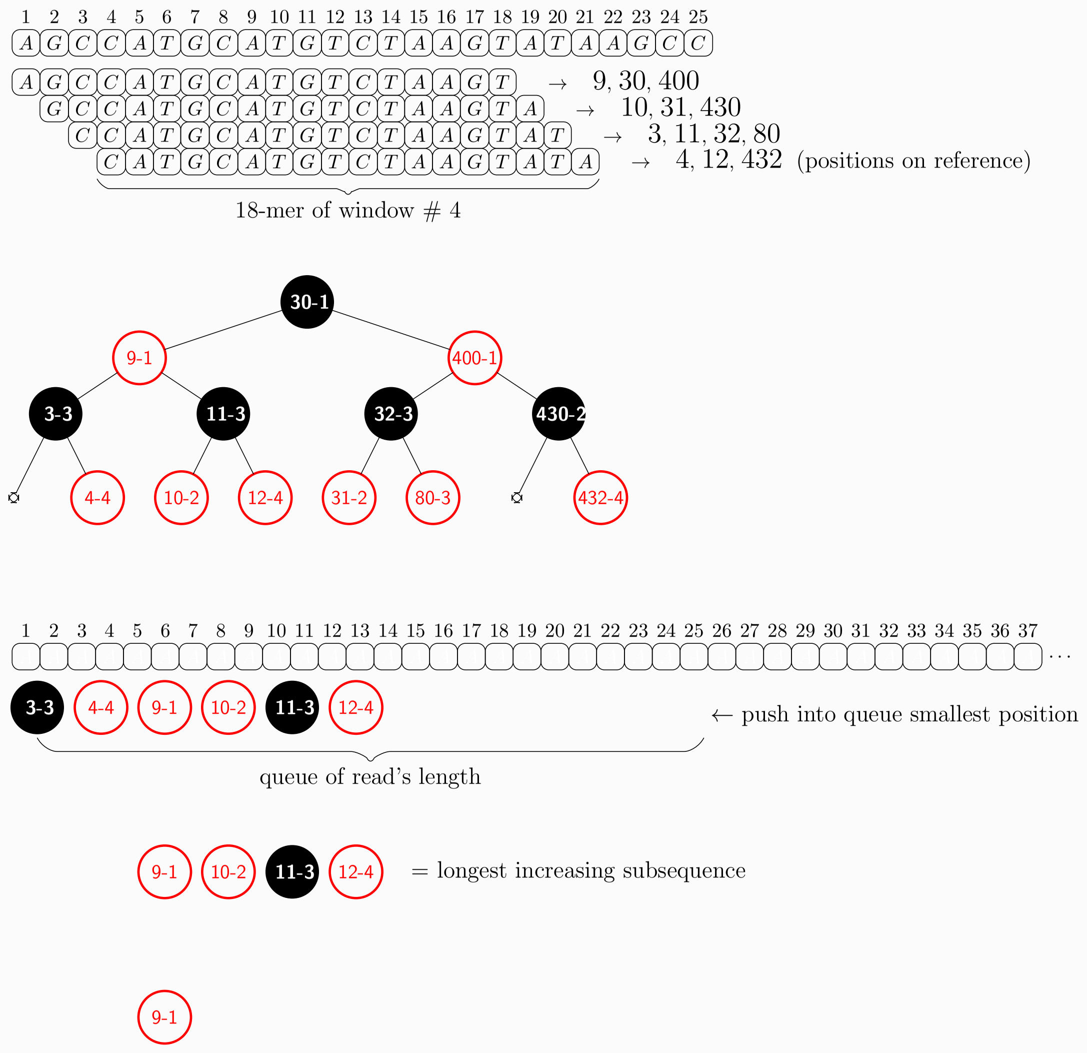
Step 1. For each k-mer on the read, collect the list of positions where it appears on the reference

Step 2. Sort all lists by position on the reference using quicksort, or a self-balancing tree, or a min-heap array (the key is the position on reference, the value is the position on the read)

Step 3. Using a double-ended queue of the length of the read, push into the queue the smallest reference positions fitting within the range of read's length

Step 4. If the number of elements in the queue is greater than threshold r, find the **longest-increasing-subsequence** of the corresponding read k-mer positions

Step 5. Apply **Smith-Waterman** alignment starting near the first position of the LIS



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
A G C C A T G C A T G T C T A A G T A T A A G C C
A G C C A T G C A T G T C T A A G T → 9, 30, 400
G C C A T G C A T G T C T A A G T A → 10, 31, 430
C C A T G C A T G T C T A A G T A T → 3, 11, 32, 80
C A T G C A T G T C T A A G T A T A → 4, 12, 432 (positions on reference)
18-mer of window # 4
30-1
9-1
400-1
3-3
11-3
32-3
430-2
4-4
10-2
12-4
31-2
80-3
432-4
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 37 ...
3-3 4-4 9-1 10-2 11-3 12-4
← push into queue smallest position
queue of read's length
9-1 10-2 11-3 12-4 = longest increasing subsequence
9-1

Chapter 3

SortMeRNA: a filter for metatranscriptomic data

Contents

In this chapter, we present a first example for an application of the approximate seeding framework introduced in Chapter 2. We designed an efficient filter to rapidly sort through millions of reads generated by metatranscriptomic sequencing projects and identify the ribosomal RNA fragments within. This classification step is a prerequisite to any further bioinformatic analysis.

The method is implemented in a software called SortMeRNA, that was released in October 2012 and published in the journal *Bioinformatics* [Kopylova et al., 2012]. It is now used in production by Genoscope (French National Center for Sequencing) to process data from Tara Oceans. It has also been integrated in two published computational pipelines [Leimena et al., 2013, Krohn-Molt et al., 2013] and received excellent feedback from multiple research laboratories worldwide[3].

This chapter is organized as follows. In Section 3.1, we present the biological context of metatranscriptome sequencing projects and the computational challenges posed by this new sort

[3]Umeå University (Sweden), Leibniz Institute DSMZ (Germany), NGS department of Campus Science Support Facilities GmbH (Austria), Oxford Centre for Integrative Systems Biology (Great Britain), Laboratoire d'Ecologie Alpine (Grenoble), PRABI (Lyon), Wageningen University (Netherlands), SciLifeLab (Stockholm), SCELSE (Singapore), ..

of data. In Section 3.2, we describe the algorithm behind SortMeRNA. In Section 3.3, we provide performance results on simulated and real data, with a comprehensive comparison with other software programs. In Section 3.4, we conclude the chapter with a discussion on the overall results.

3.1 Application context: metatranscriptomics analysis

The transcriptome of an organism consists of the set of total RNA, which regularly varies and harmonizes with external environmental conditions and the *meta*transcriptome is an ensemble of all RNA molecules found within a microbial community. Metatranscriptomic profiling via next-generation sequencing provides an authentic representation of species richness within a community at the time of the sampling. It becomes particularly important for samples which cannot be cultivated outside their native environment.

The initial challenge of metatranscriptomic sequenced data analysis is to sort apart the RNA fragments based on their biological significance. Phylogenetic structure of a community is primarily established using the 16S and 18S ribosomal RNA (rRNA) genes, which remain highly conserved among different species of bacteria, archaea and eukarya [Janda and Abbott, 2007]. Other inquiries can be made regarding the functionality of a community by studying the messenger RNA (mRNA) of the actively transcribed genes. Thus, it is of primary interest to sort out the rRNA from the mRNA in the total RNA for answering the most basic questions regarding the species composition, gene regulation and protein information of a metatranscriptome. Subject to prokaryotic or eukaryotic cells, the rRNA content can represent up to 80-85% of total RNA and the protein coding mRNA as little as 1-6% [Sorek and Cossart, 2010]. If one wants to focus exclusively on mRNA, there exist prior-to-sequencing methods to help isolate and enrich mRNA from the total RNA. These methods focus on the depletion of rRNA with technologies such as subtractive hybridization (Life Technologies), exonuclease digestion [Boissinot et al., 2007] and the duplex-specific nuclease treatment (DSN) [Yi et al., 2011]. However, even selective removal of rRNA genes does not guarantee definite depletion, and some rRNA genes can still remain in the metatranscriptomic sample. Alternatively, if the goal is to sort apart RNA without any loss of information, in the case of studying rRNA for species identification, non-invasive methods for separating RNA are also preferred. So there is a need for computational tools able to efficiently catalog families of rRNA sequences in metatranscriptomis datasets.

3.2 SortMeRNA

The goal of SortMeRNA is to solve the following problem: Given a large set of metatranscriptomic reads, how to rapidly isolate the reads belonging to rRNA sequences. Multiple software have been recently developed to address this issue. All of them take advantage of the fact that rRNA sequences are homologs and that the primary structure is well conserved within a biological domain (archaea, bacteria or eukarya). Their major difference is the method used to compare reads against a database of rRNA sequences. The first set of programs, Meta-RNA [Huang et al., 2009], SSU-ALIGN [Nawrocki et al., 2009] and rRNASelector [Lee et al., 2011] share a common approach to represent a rRNA database as a probabilistic model. Both Meta-RNA and rRNASelector use prebuilt Hidden Markov Models (HMM) and consequently sort reads against the database with the HMMER3 package [Eddy, 2011] whereas SSU-ALIGN uses Covariance Models to support secondary structure information. Alternative tools such as BLASTN [Altschul et al., 1990], used in numerous home-made workflows, and riboPicker [Schmieder et al., 2012] exploit a variation of the seed-and-extend strategy. The tool riboPicker implements a modified version of the Burrows-Wheeler Aligner [Li and Durbin, 2009]. In this context however, reads should be compared against large rRNA databases to achieve a good sensitivity level. In all cases, computational time is still an issue to handle large collections of reads.

3.2.1 Principle of the algorithm

SortMeRNA uses a seeding strategy, just like BLASTN and riboPicker. The novelty is that it combines two main fundamentals to achieve a high sensitivity and fast speed. First, it uses approximate seeds, instead of exact seeds as riboPicker does. We will see in Section 3.3 that it allows us to override some limitations of riboPicker on the sensitivity. Second, the rRNA database is stored in a text index. This allows us to take advantage of redundancy between homolog sequences, as HMMs do, and to build a compressed model of all rRNA sequences. This yields a significant speed up in the computational time, compared to BLASTN.

The algorithm is a straightforward application of the approximate seed framework presented in Chapter 2. Let k be the size of the seed.

1. The reference database is composed of a collection of rRNA sequences that are indexed in the look up table and the mini burst tries. A multiple sequence alignment of an rRNA database can clearly define areas of high nucleotide conservation and emphasize the evolutionary origins shared between organisms. Figure 3.1 shows such an example of different levels of conservation. Since we are interested in well-conserved regions of the rRNA sequences, before a k-mer is traversed in the mini-burst trie its prefix or suffix must exist a threshold β[4] number of times in the lookup table. This notion enforces that a read matches closely to one region in a database rather than multiple scattered ones leading to a false alignment.

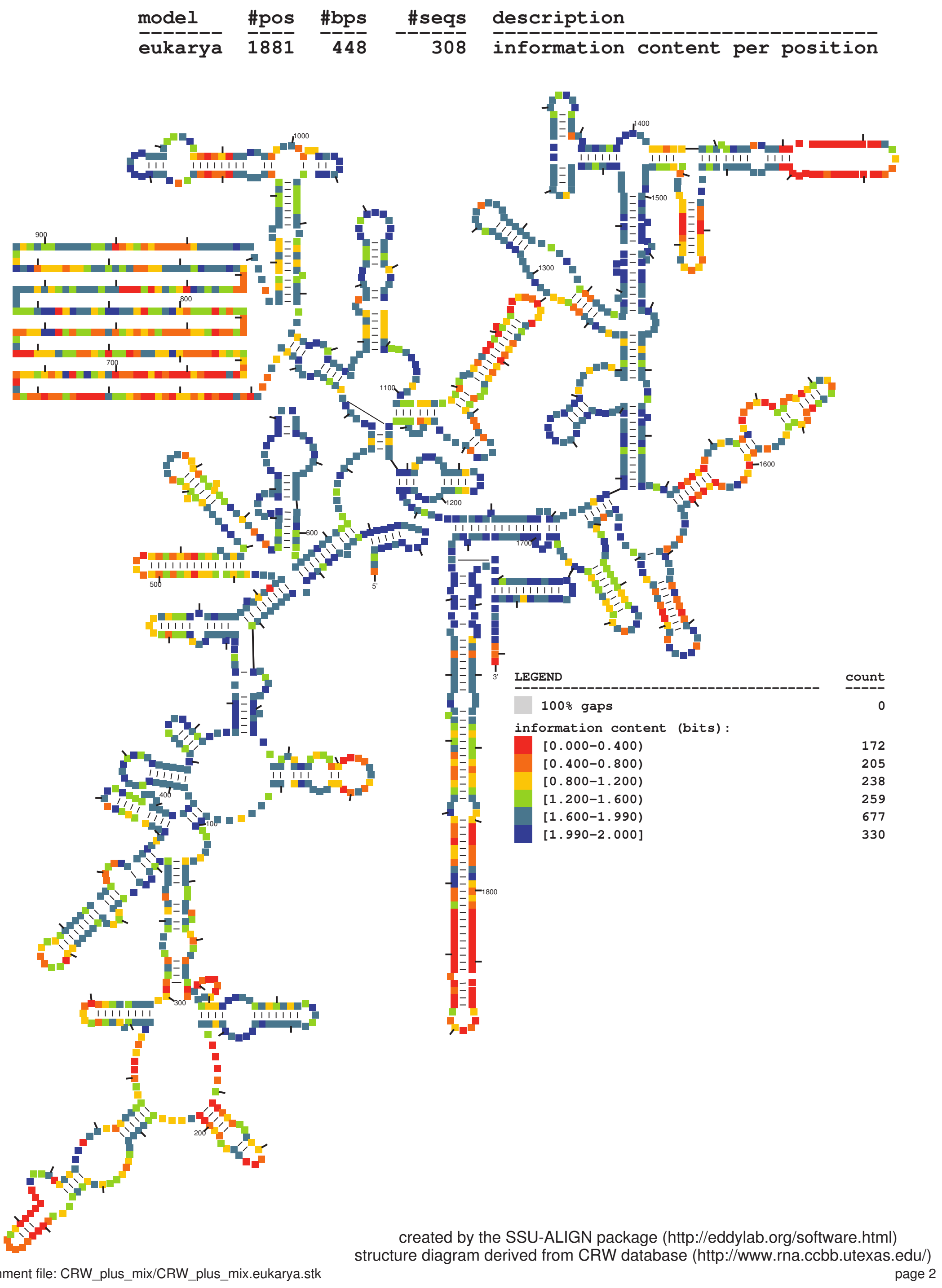
2. We scan each read with a sliding window of length k (the size of the seed), position by position, and count the number of windows present in the reference database, with up to one error. This is done with the universal Levenshtein automaton.

SortMeRNA does not implement an extension phase. The accepted reads are those which have more than a threshold ratio r of windows with a match in the database. This threshold is proportional to the length of the read.

The performances of the algorithm heavily depend on two parameters: The size k of the sliding window, and the minimal proportion r of accepted windows in a read. To find a robust choice for k and r, we ran the algorithm for several values of k and r on different rRNA databases and for several sets of reads. We discuss the choice for paramater settings in the remaining of this section.

[4]The algorithm to compute β ascertains that each read (in the set of reads provided by the user) has at least ratio r seeds, of which at least one $\frac{k}{2}$-mer has an occurrence greater than β in the rRNA database

Figure 3.1: 18S rRNA secondary structure diagram showing the high conservation (blue areas) of nucleotides in the primary structure. This diagram was generated by SSU-ALIGN for a multiple alignment of 308 eukaryotic 18S rRNA.



```
model     #pos  #bps   #seqs  description
--------  ----  ----  ------  ------------------------------------
eukarya   1881   448     308  information content per position
```

```
LEGEND                                          count
----------------------------------------------  -----
⬜ 100% gaps                                        0
information content (bits):
🟥 [0.000-0.400)                                  172
🟧 [0.400-0.800)                                  205
🟨 [0.800-1.200)                                  238
🟩 [1.200-1.600)                                  259
🟦 [1.600-1.990)                                  677
🟦 [1.990-2.000]                                  330
```

created by the SSU-ALIGN package (http://eddylab.org/software.html)
structure diagram derived from CRW database (http://www.rna.ccbb.utexas.edu/)

alignment file: CRW_plus_mix/CRW_plus_mix.eukarya.stk

page 2

3.2.2 Parameter setting

To estimate best values for k and r, we purposely designed four rRNA databases differing in terms of similarity and the rRNA subunit. We also generated several sets of simulated reads, that should be classified either as *rRNA reads*, or as *non-rRNA reads*, and for which the classification results are known.

3.2.2.1 Construction of rRNA databases

We use the SILVA databases [Pruesse et al., 2007] that administer a comprehensive set of quality checked rRNAs, including a software tool ARB [Ludwig et al., 2004] to aid with phylogenetic analyses of the data via graphical representation. From SILVA, we purposely designed four databases with distinctive features: Small 16S and large 23S subunit, varying identity percentage and from distinct phylogeny tree subparts.

Set 1 : 80% identity 16S rRNA bacteria & archaea (2262 sequences)
Set 2 : 80% identity 16S rRNA bacteria & archaea+truncated phylogeny tree (2187 sequences)
Set 3 : 95% identity 23S rRNA bacteria & archaea (1969 sequences)
Set 4 : 95% identity 23S rRNA bacteria & archaea+truncated phylogeny tree (1906 sequences)

Each database was constructed by applying the ARB package and UCLUST [Edgar, 2010] to the small 16S and large 23S subunit databases from SILVA. The procedure is as follows (see also Figure 3.2, Figure 3.3, Figure 3.4, Figure 3.5 for each Set),

1. use the ARB package to extract the phylogeny trees of 16S rRNA and 23S rRNA bacteria & archaea databases in fasta and xml formats,
2. for Set 2 and Set 4, remove a branch in the phylogenetic tree to induce missing species (see Figures 3.3 and 3.5)
3. remove all occurrences of rRNA other than 16S or 23S using description information in the xml format
4. remove long sequences (1600 for 16S rRNA and 5000 for 23S rRNA) and those having $> 1\%$ of ambiguous N's,
5. apply UCLUST on the filtered set of rRNAs to create representative 16S and 23S rRNA databases with identity $x\%$.

   The identity percentage x refers to the definition of clusters used by UCLUST: Each cluster is defined by a representative sequence, and each sequence in a cluster matches the representative sequence according to the identity threshold. Finally, each database is constituted by the set of representative sequences. By construction, every pair of sequences in the representative database has an identity percentage lower than the threshold x.

3.2.2.2 Simulated reads

We generated simulated reads to be able to estimate the selectivity and the sensitivity of SortMeRNA with varying parameter values. For that, we applied MetaSim [Richter et al., 2008] using provided error models: Roche 454 and Illumina. MetaSim's maximum length error model for the

Figure 3.2: Construction of Set 1 – 16S rRNA database with 80% identity

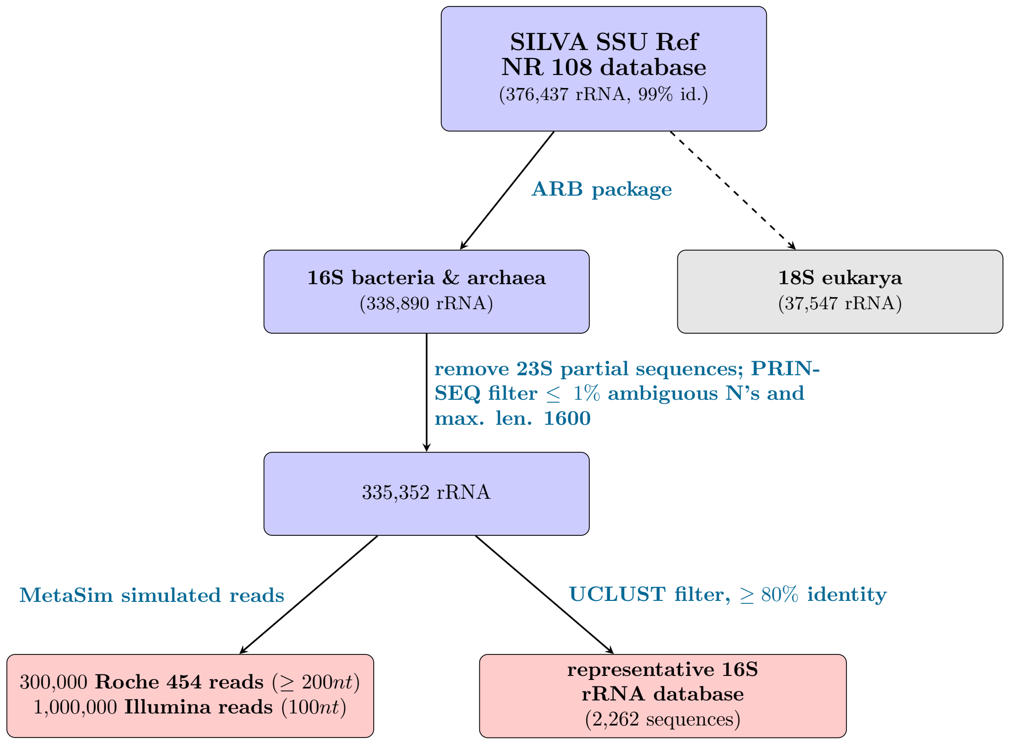

SILVA SSU Ref
NR 108 database
(376,437 rRNA, 99% id.)
ARB package
16S bacteria & archaea
(338,890 rRNA)
18S eukarya
(37,547 rRNA)
remove 23S partial sequences; PRINSEQ filter ≤ 1% ambiguous N's and max. len. 1600
335,352 rRNA
MetaSim simulated reads
UCLUST filter, ≥ 80% identity
300,000 Roche 454 reads (≥ 200nt)
1,000,000 Illumina reads (100nt)
representative 16S rRNA database
(2,262 sequences)

Figure 3.3: Construction of Set 2 – 16S rRNA database with 80% identity + truncated phylo. tree

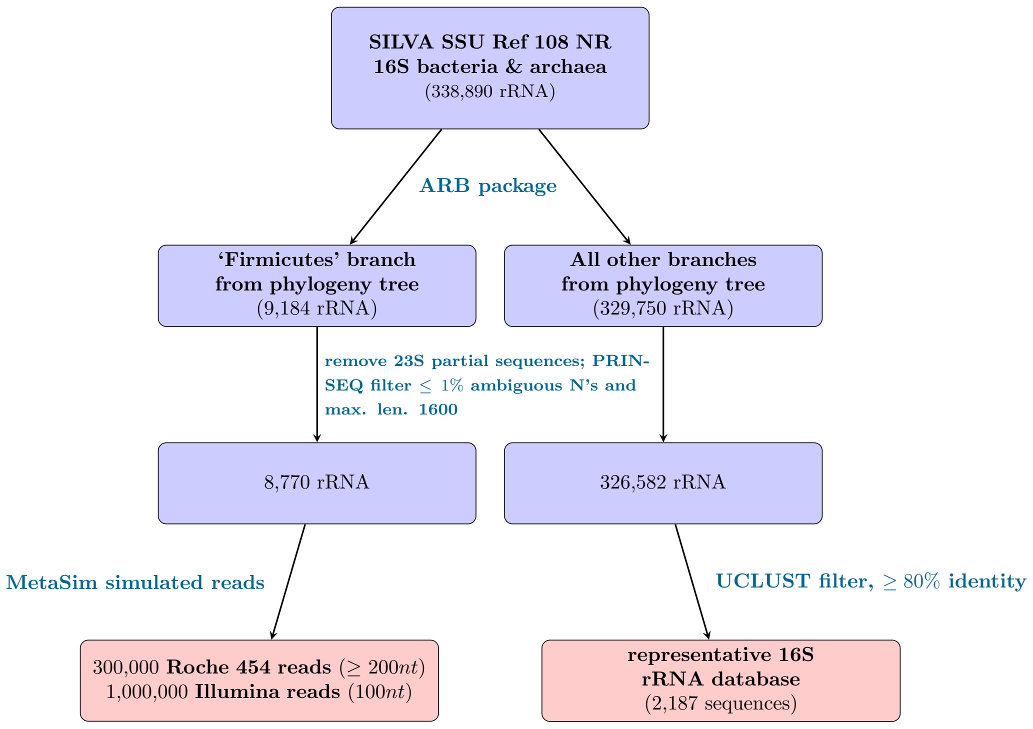

SILVA SSU Ref 108 NR
16S bacteria & archaea
(338,890 rRNA)
ARB package
'Firmicutes' branch from phylogeny tree
(9,184 rRNA)
All other branches from phylogeny tree
(329,750 rRNA)
remove 23S partial sequences; PRINSEQ filter ≤ 1% ambiguous N's and max. len. 1600
8,770 rRNA
326,582 rRNA
MetaSim simulated reads
UCLUST filter, ≥ 80% identity
300,000 Roche 454 reads (≥ 200nt)
1,000,000 Illumina reads (100nt)
representative 16S rRNA database
(2,187 sequences)

Figure 3.4: Construction of Set 3 – 23S rRNA database with 95% identity

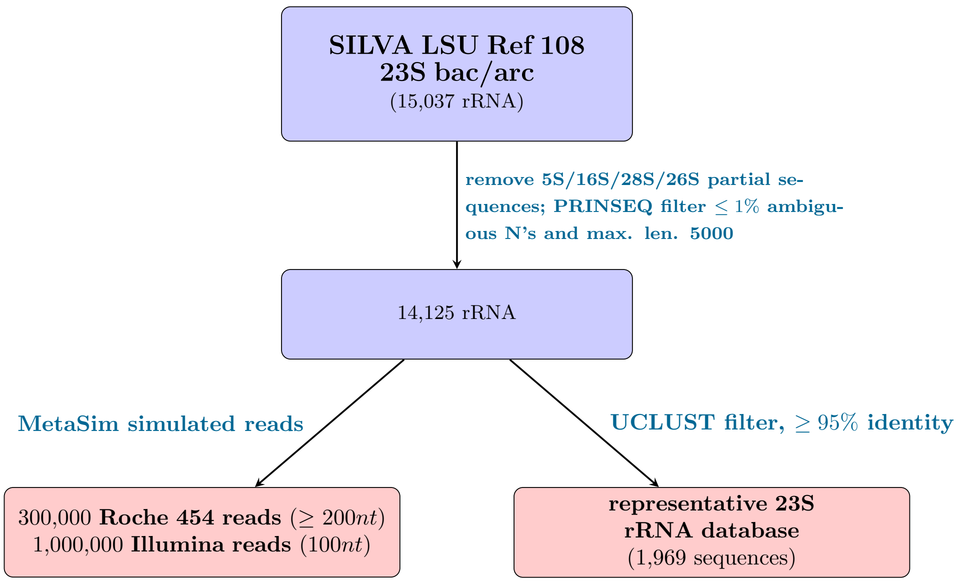
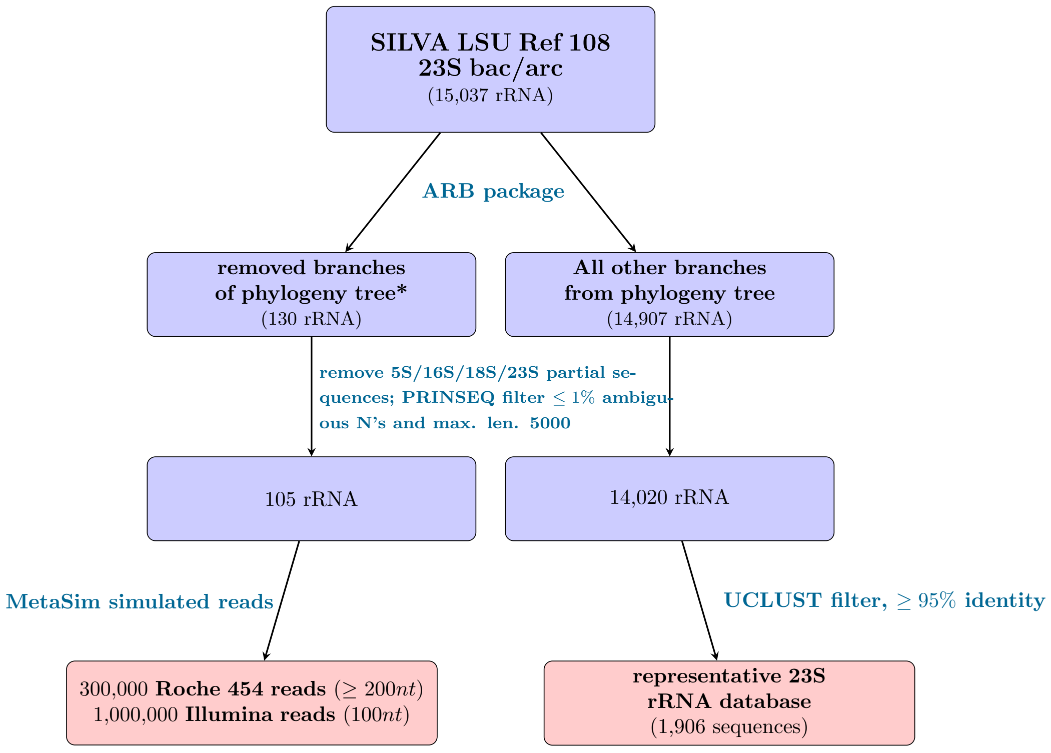

SILVA LSU Ref 108
23S bac/arc
(15,037 rRNA)
remove 5S/16S/28S/26S partial sequences; PRINSEQ filter ≤ 1% ambiguous N's and max. len. 5000
14,125 rRNA
MetaSim simulated reads
UCLUST filter, ≥ 95% identity
300,000 Roche 454 reads (≥ 200nt)
1,000,000 Illumina reads (100nt)
representative 23S rRNA database
(1,969 sequences)

Figure 3.5: Construction of Set 4 – 23S rRNA database with 95% identity + truncated phylogenetic tree. Section of phylogenetic tree: 36 Planctomycetes, 14 Fibrobacteres, 44 Verrucomicrobia, 21 Chloroflexi_1, 6 Candidate division TM7, and 9 Lentisphaerae.


SILVA LSU Ref 108
23S bac/arc
(15,037 rRNA)
ARB package
removed branches of phylogeny tree*
(130 rRNA)
All other branches from phylogeny tree
(14,907 rRNA)
remove 5S/16S/18S/23S partial sequences; PRINSEQ filter ≤ 1% ambiguous N's and max. len. 5000
105 rRNA
14,020 rRNA
MetaSim simulated reads
UCLUST filter, ≥ 95% identity
300,000 Roche 454 reads (≥ 200nt)
1,000,000 Illumina reads (100nt)
representative 23S rRNA database
(1,906 sequences)

Illumina technology is 80nt, in order to adapt this model for 100nt the last probability value of the 80th position was extended by 20nt. In practice, MetaSim had simulated Roche 454 reads with 2.8-3% sequencing error rate of which approximately 79% were insertions and 21% deletions. Additionally, for Illumina reads, the sequencing error rate was 1.2% of which 100% were substitutions.

Generation of Roche 454 and Illumina rRNA reads. We started from sequences of the SILVA database that have not been already selected in the representative sets (Set 1 to Set 4).

1. apply MetaSim on the filtered set of rRNAs of SILVA not belonging to the representative databases to create 300,000 Roche 454 reads of $\geq 200nt$ and 1,000,000 Illumina reads of 100nt.

   In the MetaSim simulator settings, the parameters 'Mean' (mean length of clone) and 'Second Parameter' (standard deviation of clone length) as defined in the user manual, were set to 1000 and 100 respectively. The default values are 2000 and 200, however since many rRNA sequences are shorter than 2000 nt, we reduced this value to 1000 and the standard deviation to 100.

2. filter out 'circular' reads produced by MetaSim, approximately 10% of the reads.

3. filter out reads shorter than 200nt (only for Roche 454) and reads with $> 1\%$ of ambiguous character N (both Roche 454 & Illumina).

Generation of Roche 454 and Illumina non-rRNA reads We started from the NCBI complete bacterial genomes: ftp://ftp.ncbi.nlm.nih.gov/genomes/Bacteria/

1. find all locations of rRNAs on the complete NCBI bacterial genomes in the Genbank file format,

2. mask the locations of rRNA in the Fasta file format by a contiguous sequence of N's (+150nt to cover misannotation)

3. run MetaSim on the NCBI complete bacterial genomes with masked rRNAs to create 1,000,000 Roche 454 reads of $\geq 200nt$ and 1,000,000 Illumina reads of 100nt,

4. filter out 'circular' reads produced by MetaSim, approximately 10% of the reads.

   In the MetaSim simulator settings, the parameters 'Mean' (mean length of clone) and 'Second Parameter' (standard deviation of clone length) as defined in the user manual, were set to 2000 and 200 respectively. These default values work well since the average length of a bacterial genome exceeds 2000 nt.

5. filter out reads shorter than 200nt (only for Roche 454) and reads with $> 1\%$ of ambiguous character N (both Roche 454 & Illumina).

Note that some rRNA genes are not annotated and therefore missed during the masking process. This explains that there may be some rRNA fragments in the generated reads, which will impact the selectivity for all programs.

3.2.2.3 Choice of values for k and r

To estimate robust values for k and r, we ran SortMeRNA on Set 1 to Set 4 and varied the parameters $k \in [14, 16, 18, 20]$ and $r \in [0.05, 0.10, 0.15, \ldots, 0.95]$. The acceptance rule is as follows. Let ℓ be the length of a read, and ϵ the number of accepted windows for this read. A read is classified as rRNA if the number of accepted windows divided by the number of total full windows on a read is greater or equal to r,

$$\frac{\epsilon}{\ell - k + 1} \geq r.$$

The results for Roche 454 reads are demonstrated in Figure 3.6 and those for Illumina reads in Figure 3.7. The main conclusion is that $k = 18$, $r = 0.15$ for Roche 454 reads and $k = 18$, $r = 0.25$ for Illumina reads give best sensitivity/selectivity balance for all rRNA databases. Moreover, as shown in the Matthews correlation coefficient tables, varying r within short ranges does not significantly affect the results. We use these values as default settings in all subsequent analyses of Section 3.3.

3.2.3 Availability

SortMeRNA is written in C++ and freely distributed under the GPL license as a stand-alone version or as a Galaxy wrapper. Galaxy is an open, web-based platform that provides users with a graphical workflow management system, and emphasizes accessibility, reproducibility and transparency [Goecks et al., 2010]. Both distributions, including the user manual with installation instructions, can be downloaded from `http://bioinfo.lifl.fr/RNA/sortmerna/`.

The software uses OpenMP functions to parallelize filtering of the reads. The input criteria are a fasta/fastq file of letter space reads produced by Roche 454 or Illumina technologies, and a fasta file of rRNA sequences. There are eight rRNA databases included in the software package covering the small (16S/18S), large (23S/28S) and 5/5.8S ribosomal subunit rRNAs, which were all derived from the SILVA and RFAM databases. Additionally, the user can work with their own RNA databases.

SortMeRNA supports multi-threading and has been tested on Linux (Ubuntu, Fedora, CentOS and Debian) and Mac OS 10.6.8 systems. For compilation, a g++ compiler version 4.3 or higher is required.

3.3 Performance results

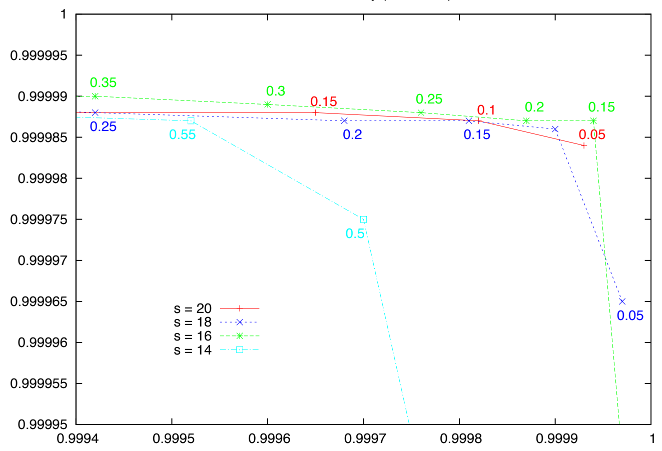
The performance of SortMeRNA was measured in terms of sensitivity, selectivity and real-data analysis compared to the previously mentioned software SSU-ALIGN, Meta-RNA, rRNASelector, riboPicker and BLASTN.

For evaluating all software on an equal basis, another two databases were created: One for 16S rRNA, and one for 23S rRNA.

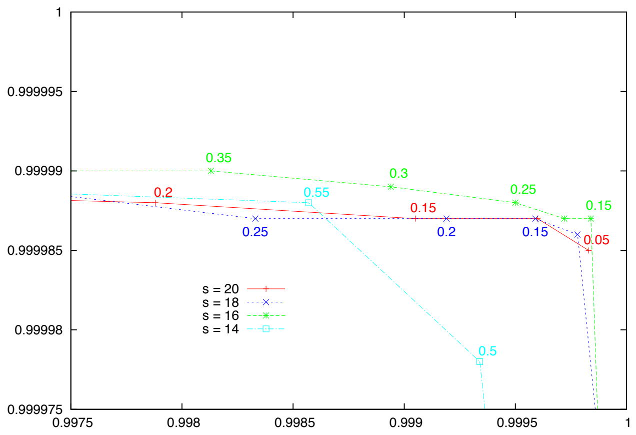
Set 5 : 85% identity 16S rRNA bacteria & archaea (7659 rRNA), see Figure 3.8
Set 6 : 98% identity 23S rRNA bacteria & archaea (2811 rRNA), see Figure 3.9

The 16S rRNA database was used by SortMeRNA, riboPicker, BLASTN and SSU-ALIGN in Test 1 and Test 3, and the 23S rRNA database by SortMeRNA, riboPicker and BLASTN in

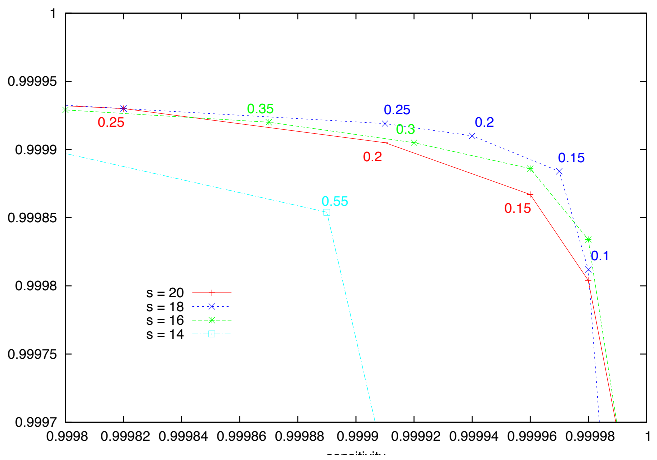
Figure 3.6: SortMeRNA results by varying parameters k (length of the sliding window) and r (ratio of accepted windows) on Set 1 to Set 4 for **Roche 454** simulated reads. For each graph, the horizontal axis is for the sensitivity, and the vertical axis is for the selectivity. Each curve corresponds to a different value for k: $k = 14, 16, 18, 20$. Each dot on a curve corresponds to a different value for r. Below: Matthews correlation coefficients for $k = 18$ and various values of r.



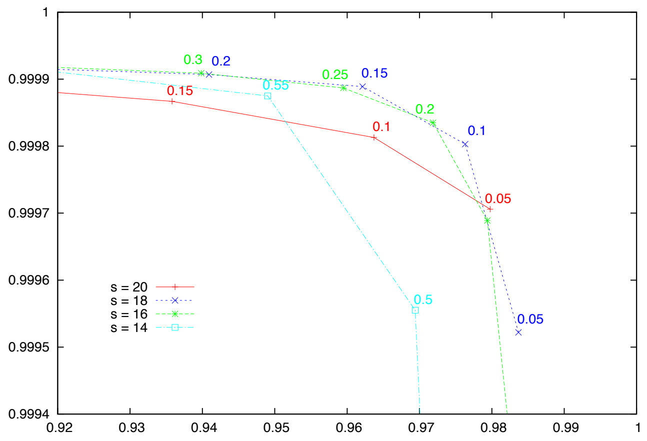
Set 1: 80% identity 16S rRNA



Set 2: 80% identity 16S rRNA, truncated tree



Set 3: 95% identity 23S rRNA

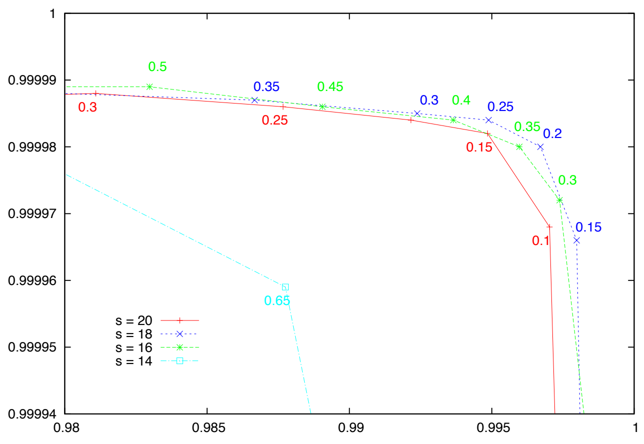


Set 4: 95% identity 23S rRNA, truncated tree

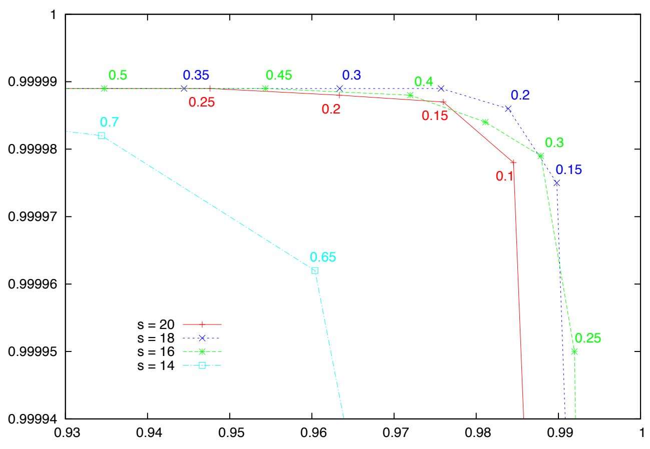
Matthews correlation coefficient

r	0.05	0.1	0.15	0.2	0.25	0.3	0.35
Set 1	0.9999	0.9999	0.99985	0.99976	0.99959	0.99933	0.99895
Set 2	0.99987	0.99982	0.99971	0.99945	0.99889	0.99773	0.99536
Set 3	0.99895	0.99958	0.99973	0.99977	0.99977	0.99973	0.99970
Set 4	0.98834	0.98417	0.97512	0.96131	0.93811	0.89803	0.84026

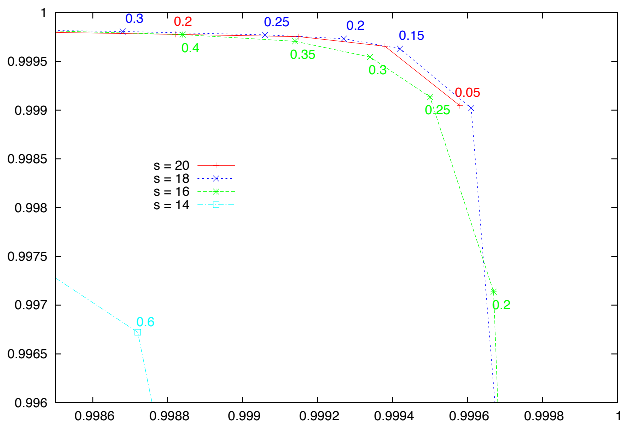
Figure 3.7: SortMeRNA results by varying parameters k (length of the sliding window) and r (ratio of accepted windows) on Set 1 to Set 4 for **Illumina** simulated reads. For each graph, the horizontal axis is for the sensitivity, and the vertical axis is for the selectivity. Each curve corresponds to a different value for k: $s = 14, 16, 18, 20$. Each dot on a curve corresponds to a different value for r. Below: Matthews correlation coefficients for $k = 18$ and various values of r.



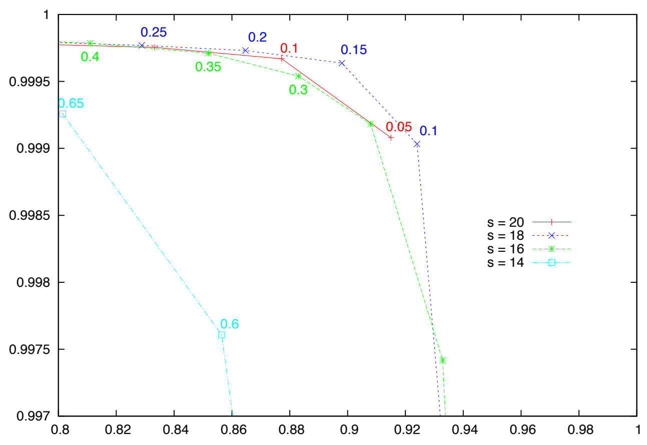
Set 1: 80% identity 16S rRNA



Set 2: 80% identity 16S rRNA, truncated tree



Set 3: 95% identity 23S rRNA



Set 4: 95% identity 23S rRNA, truncated tree

Matthews correlation coefficient

r	0.05	0.1	0.15	0.2	0.25	0.3	0.35
Set 1	0.99598	0.99856	0.99795	0.99670	0.99489	0.99240	0.98674
Set 2	0.99366	0.99395	0.98982	0.98401	0.97600	0.96401	0.94588
Set 3	0.99306	0.99863	0.99905	0.99901	0.99883	0.99849	0.99792
Set 4	0.94040	0.92567	0.90233	0.87242	0.84096	0.80262	0.75228

Test 2 and Test 4. SSU-ALIGN was written for aligning small ribosomal subunits and does not provide models for 23S rRNA. The tool riboPicker was also tested with a more comprehensive database made available from their website: All 16S and 23S rRNA sequences taken from SILVA, RDP-II, Greengenes, NCBI archaeal and bacterial genomes, and HMP (3,232,371 16S and 19,602 23S unique sequences). The results for this larger database are indicated by riboPicker* in the subsequent tables and figures. For Meta-RNA and rRNASelector, we used the HMMs provided with the software.

All tests were performed on an Intel(R) Xeon(R) CPU W3520 2.67GHz machine with L1 cache size of 32 KB, L2 cache size of 256 KB and L3 cache size of 8192 KB. Since riboPicker and SSU-ALIGN do not provide a direct option for multi-threading, all tests were carried out using one thread.

3.3.1 Test 1: simulated 16S rRNA reads

Sensitivity for 16S rRNA. 300,000 Roche 454 and 1,000,000 Illumina 16S rRNA reads were simulated in the same manner as described in Section 3.2.2.2. The performance results can be viewed in Table 3.1. All software programs except riboPicker and SSU-ALIGN have a sensitivity level higher than 97%, and even higher than 99% for BLASTN and SortMeRNA. The sensitivity for riboPicker is very low (56%) because BWA-SW works well with error rates 2%-3% for 100-200nt reads, and loses sensitivity for new species. As expected, the sensitivity increases with a larger database (indicated riboPicker*). Considering the computation time, SortMeRNA runs in less than 2 minutes, or 72x faster than the next fastest tool with proportionate sensitivity (Meta-RNA). Note also that BLASTN executes at a very slow speed (several hours), because reads should be compared against all of sequences in the representative database.

Selectivity for 16S rRNA. 1,000,000 Roche 454 and 1,000,000 Illumina non-16S rRNA reads were simulated in the same manner as described in Section 3.2.2.2. The performance results can be viewed in Table 3.2. All programs have a selectivity level higher than 99.98%. The number of false positives for the HMM-based programs remains comparable to SortMeRNA for both Illumina and Roche 454 reads. The difference in the simulated data results between Meta-RNA and rRNASelector can be attributed to the number of bacteria vs. archaea rRNA sequences used in the construction of the HMMs, as well as additional parameter settings in rRNASelector. riboPicker* and BLASTN show the lowest selectivity. Concerning the running time, the order of the fastest programs is rRNASelector, Meta-RNA and SortMeRNA. Both rRNASelector and Meta-RNA use the HMMER3 package, which applies a pre-filter to quickly reject sequences which would score very low in the HMM. This acceleration heuristic gives these programs a competitive advantage on the artificial dataset for selectivity where all of the sequences are negative.

3.3.2 Test 2: simulated 23S rRNA reads

Similarly, we generated rRNA and non-rRNA reads for 23S rRNAs using the same protocol, as described in Section 3.2.2.2. Results are analogous as those of Test 1 in terms of accuracy and running time. They can be found in Table 3.3 and Table 3.4.

Figure 3.8: Construction of Set 5 – representative 16S rRNA database with 85% identity

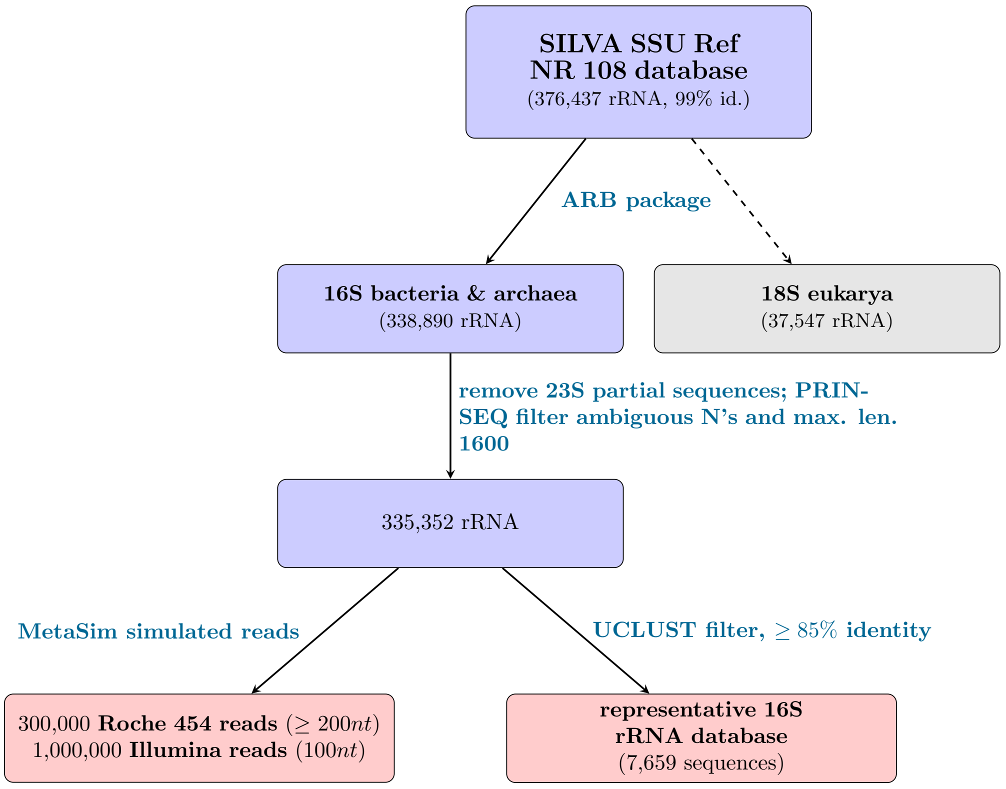

SILVA SSU Ref NR 108 database
(376,437 rRNA, 99% id.)
ARB package
16S bacteria & archaea
(338,890 rRNA)
18S eukarya
(37,547 rRNA)
remove 23S partial sequences; PRINSEQ filter ambiguous N's and max. len. 1600
335,352 rRNA
MetaSim simulated reads
UCLUST filter, ≥ 85% identity
300,000 Roche 454 reads (≥ 200nt)
1,000,000 Illumina reads (100nt)
representative 16S rRNA database
(7,659 sequences)

Figure 3.9: Construction of Set 6 – representative 23S rRNA database with 98% identity

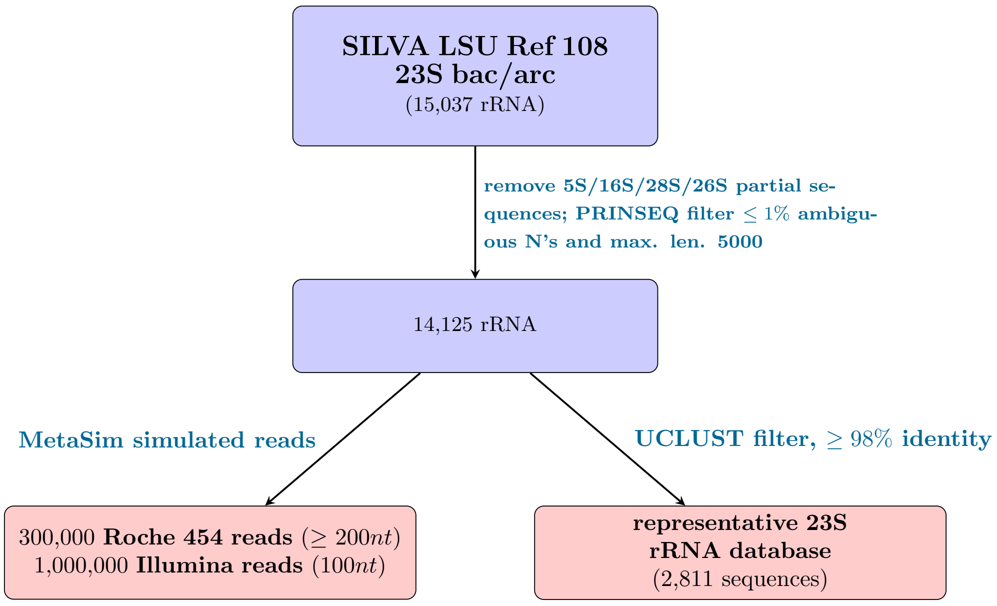

SILVA LSU Ref 108
23S bac/arc
(15,037 rRNA)
remove 5S/16S/28S/26S partial sequences; PRINSEQ filter ≤ 1% ambiguous N's and max. len. 5000
14,125 rRNA
MetaSim simulated reads
UCLUST filter, ≥ 98% identity
300,000 Roche 454 reads (≥ 200nt)
1,000,000 Illumina reads (100nt)
representative 23S rRNA database
(2,811 sequences)

Table 3.1: TEST 1, SENSITIVITY. 1,000,000 of MetaSim simulated Illumina (100*nt*) and 300,000 Roche 454 ($\geq$ 200*nt*) rRNA reads against a representative 16S rRNA database of 7,659 sequences.

	Illumina				Roche 454			
	rRNA	run time (hrs:min)	memory (%)	sensitivity (%)	rRNA	run time	memory (%)	sensitivity (%)
SortMeRNA	998615	0:02	8.5	99.861	299979	0:02	6.3	99.993
riboPicker	558607	0:19	6.8	55.860	123024	0:19	5.6	41.008
riboPicker *	999941	6:33	35.3	99.994	299999	9:00	34	99.999
BLASTN	995322	23:52	3.0	99.532	299978	18:35	1.4	99.992
Meta-RNA	983332	2:00	33.3	98.333	299980	1:57	12.9	99.993
rRNASelector	974118	1:47	17.4	97.411	299976	2:00	7	99.992
SSU-ALIGN	971221	6:49	0.1	97.122	299902	5:50	0.1	99.967

Table 3.2: TEST 1, SELECTIVITY. 1,000,000 of MetaSim simulated Illumina (100*nt*) and 1,000,000 Roche 454 ($\geq$ 200*nt*) non-rRNA reads against a representative 16S rRNA database of 7,659 sequences.

	Illumina				Roche 454			
	rRNA	run time (hrs:min)	memory (%)	sensitivity (%)	rRNA	run time (hrs:min)	memory (%)	sensitivity (%)
SortMeRNA	17	0:02	7.6	99.9983	13	0:04	10.2	99.9987
riboPicker	7	0:10	6.7	99.9993	3	0:30	16.8	99.9997
riboPicker *	158	0:57	35.1	99.9842	53	2:43	45.2	99.9947
BLASTN	33	0:14	0.3	99.9967	33	0:16	0.3	99.9967
Meta-RNA	11	0:02	0.1	99.9989	11	0:04	0.2	99.9989
rRNASelector	10	0:01	0.1	99.9990	11	0:03	0.2	99.9989
SSU-ALIGN	8	3:51	0.1	99.9992	11	10:30	0.1	99.9989

Table 3.3: TEST 2, SENSITIVITY. 1,000,000 of MetaSim simulated Illumina (100*nt*) and 300,000 Roche 454 ($\geq$ 200*nt*) **rRNA** reads against a representative 98% identity 23S rRNA database of 2,811 sequences.

	Illumina				Roche 454			
	rRNA	run time (hrs:min)	memory (%)	sensitivity (%)	rRNA	run time (hrs:min)	memory (%)	sensitivity (%)
SortMeRNA	999909	0:01	7.2	99.909	300000	0:01	4.7	100
riboPicker	659494	0:19	6.7	65.949	213989	0:21	5.6	71.329
riboPicker *	986917	1:26	8.4	98.691	296584	1:46	7.3	98.861
BLASTN	999549	15:10	2.7	99.954	299999	11:25	1.3	99.999
Meta-RNA	936314	4:25	31.8	93.631	298918	4:29	13.1	99.639
rRNASelector	908344	4:7	16.4	90.834	298733	4:37	7.2	99.577

* Searching through all 23S rRNA databases provided by SILVA (only 23S), NCBI archaeal and bacterial genomes, and HMP.

Table 3.4: TEST 2, SELECTIVITY. 1,000,000 of MetaSim simulated Illumina (100*nt*) and 300,000 Roche 454 ($\geq$ 200*nt*) **non-rRNA** reads against a representative 98% identity 23S rRNA database of 2,811 sequences

	Illumina				Roche 454			
	rRNA	run time (hrs:min)	memory (%)	sensitivity (%)	rRNA	run time (hrs:min)	memory (%)	sensitivity (%)
SortMeRNA	243^+	0:01	6.3	99.9757	112^+	0:03	8.9	99.9888
riboPicker	39^+	0:10	6.6	99.9961	24^+	0:32	16.7	99.9976
riboPicker *	103	0:30	8.2	99.9897	54	1:32	18.3	99.9946
BLASTN	310^+	0:26	0.3	99.9690	571^+	0:26	0.2	99.9429
Meta-RNA	36	0:03	0.1	99.9964	29	0:06	0.3	99.9971
rRNASelector	34	0:02	0.1	99.9966	29	0:06	0.3	99.9971

* Searching through all 23S rRNA databases provided by SILVA (only 23S), NCBI archaeal and bacterial genomes, and HMP. $^+$ SortMeRNA, riboPicker and BLASTN use the same database (Set 6). riboPicker* searches through a database with 19,602 23S rRNA sequences, and both Meta-RNA and rRNASelector use prebuilt HMM models. For SortMeRNA, 82% of the 243 Illumina reads and 100% of the 112 Roche 454 reads are in common with the 310 and 571 reads classified by BLASTN. For riboPicker, 97% of the 39 Illumina reads and 100% of the 24 Roche 454 reads are in common with the 310 and 571 reads classified by BLASTN. The majority of these reads map to mRNA. Further investigation showed that due to misannotation, the database for Set 6 was contaminated with several mRNA, and hence the classified reads were correctly spotted in the database.

3.3.3 Test 3: photosynthetic microbial community

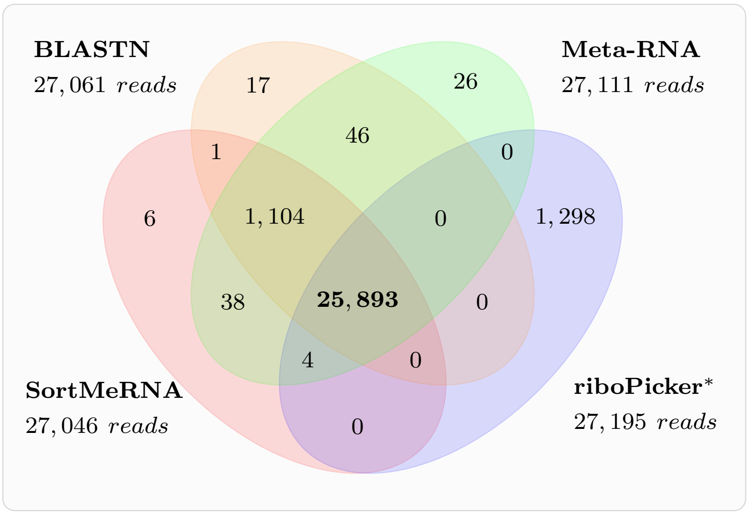
The metatranscriptomic dataset SRR106861 of a photosynthetic microbial community from 454 sequencing was downloaded from the NCBI Sequence Read Archive. We filtered this read set to remove any bias caused by shorter reads (<200*nt*), as well as low-quality reads which have more than 1% of the ambiguous character N. The reason for eliminating shorter reads is that riboPicker, Meta-RNA, rRNASelector and SSU-ALIGN require longer reads to achieve a higher sensitivity. In general, this length is suggested to be $\geq$ 200*nt* for Roche 454 reads.

The results can be viewed in Table 3.5, and the overlap of the results between tools in a Venn diagram displayed in Figure 3.10. The results obtained with SortMeRNA adhere to the accuracy of the HMM-based programs and are computed in a fraction of the time. riboPicker finds only a subpart of all potential rRNAs, which confirms its low sensitivity for small databases. The majority of 16S rRNA reads found only by riboPicker* (1,298) map to mRNA.

Table 3.5: TEST 3: Runtime for the SRR106861 metatranscriptome of 105,873 reads against a 16S rRNA database of 7,659 sequences.

	rRNA	run time (hrs:min)	memory (%)
SortMeRNA	27046	< 0:01	4.8
riboPicker	11389	0:04	2.3
riboPicker *	27195	0:39	30.8
BLASTN	27061	1:29	0.6
Meta-RNA	27111	0:11	1.8
rRNASelector	27085	0:11	0.8

Figure 3.10: TEST 3: Venn diagram for reads classified as 16S rRNA by BLASTN, Meta-RNA, SortMeRNA and riboPicker* in the SRR106861 metatranscriptome.



3.3.4 Test 4: tidal salt marsh creek

Similarly, we retrieved the dataset SRR013513 of a tidal salt marsh creek from 454 sequencing from the NCBI Sequence Read Archive. We removed low quality and short reads using the same cleaning procedure as in Test 3.

Results are available in Table 3.6 and Figure 3.11. Approximately 99% of the excess reads of Meta-RNA (12,112) and rRNASelector likewise map to 28S, along with 83% of the (624) reads found only by BLASTN and Meta-RNA. The (537) reads found only by BLASTN map to 16S rRNA, ncRNA and mRNA.

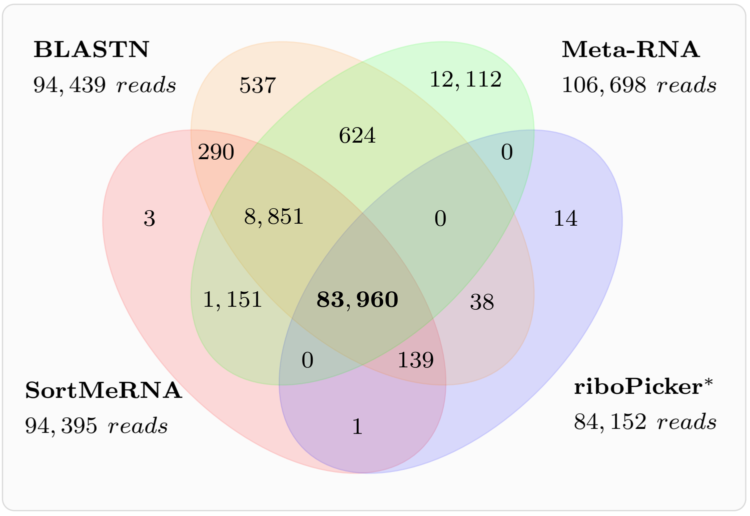
Table 3.6: TEST 4: Runtime for the SRR013513 metatranscriptome of 207,368 reads against a 23S rRNA database of 2,811 sequences.

	rRNA	run time (hrs:min)	memory (%)
SortMeRNA	94395	< 0:01	3.8
riboPicker	71937	0:10	3.9
riboPicker *	84152	0:36	5.5
BLASTN	94439	3:42	0.9
Meta-RNA	106698	1:33	4.8
rRNASelector	107900	1:36	3

3.4 Discussion

SortMeRNA has shown to be a rapid and efficient filter which can sort a large set of metatranscriptomic reads with high accuracy comparable to the HMM-based programs and a significantly lower running time. SortMeRNA implements our approximate seeds and this important charac-

Figure 3.11: TEST 4: Venn diagram for reads classified as 23S rRNA by BLASTN, Meta-RNA, SortMeRNA and riboPicker* in the SRR013513 metatranscriptome.



teristic renders the algorithm robust to errors of different types of sequencers while providing the ability to discover new rRNA sequences from unknown species. The method used by the algorithm is universal and flexible. The database can be constructed on any family of sequences that show some conservation at the nucleic acid level, which can be useful for identifying other families, such as transfer RNAs, or sequencing adapters. Moreover, the algorithm does not require a multiple sequence alignment file to build the database, as HMM-based programs do, and this is an advantage when sequences are hard to align or only partial sequences are available. Another advantage of SortMeRNA is the small number of parameter settings required by the program, most of which are precomputed using statistical analysis of the rRNA database.

In [Kopylova et al., 2013], we discuss in more detail how to use SortMeRNA in a global strategy for analyzing metatranscriptomic data: How to clean the data prior to the classification, how to map and assemble the filtered reads to proceed to the reconstruction of mRNA transcripts for functional analyses and phylogenetic classification of a community using the ribosomal RNA.

Chapter 4

SortMeDNA: a genomic and metagenomic mapping tool

Contents

In this chapter, we turn our attention to another well-known sequence analysis problem, being read mapping. The development of read mapping software capable of aligning a collection of reads against a reference genome is one of the most pronounced problems in biological sequence analysis. We show how it is possible to embed our approximate seeds framework (described in Chapter 2) into a sensitive read mapper, able to deal with divergent species or reads with high error rate.

In Section 4.1, we discuss the read mapping problem in the context of traditional single-genome alignment and then extend it to metagenomic applications which have gained extreme popularity with the arrival of high-throughput sequencing technologies. In Section 4.2, we describe the algorithmic extension of using k-mer positions to locate larger regions of similarity between a read and a reference sequence and then perform Smith-Waterman alignment and statistical analysis to verify the significance of the match. In Section 4.3, we test the performance of

SortMeDNA against popular mapping tools Bowtie2, BWA-SW, SHRiMP2 and BLASTN, and demonstrate that our tool is robust for a wide range of data with minimal user intervention, while delivering fast and accurate results.

4.1 The read mapping problem

The main challenges associated with read mapping involve efficient processing of large amounts of sequenced read data and delivering robust algorithms generic to different sequencing technologies and their characteristic error types. As seen in Section 1.3.2, the nature of the reads to map depends on the sequencing technology. Illumina, 454 and Ion Torrent produce the longest reads (100-1000 bp) with the lowest error rates in one round of sequencing, whilst the single-molecule sequencing platform, PacBio (Pacific BioScience), can produce average read lengths of 4600 bp but with a much higher error rate than for other technologies (nearly 15%). Without applying heuristics, algorithms (such as SSEARCH [Smith and Waterman, 1981, Pearson, 1991]) which identify both substitution and indel errors are computationally expensive for large quantities of data. However, although heuristics can effectively speed up the algorithm, many of them impose error-free or substitution-only 'seeding' techniques for identifying short homologs prior to extending an alignment, and thus are less capable of identifying true low-complexity regions of a sequence (see Section 1.4).

Originally, the main objective of read mapping software was to quickly and accurately align low-divergent reads against a large reference genome. Some of the well known open source tools include BWA-SW [Li and Durbin, 2009], SOAP2 [Li et al., 2009] and Bowtie2 [Langmead and Salzberg, 2012], likewise the CLC Bio genomics workbench (CLC Bio) and Novoalign (Novocraft) from the commercial sector. All of these software tools have been largely optimized for genome resequencing, specifically the resequencing of the human genome using the Illumina technology (for a review see [Hatem et al., 2013]). In the last decade, the application of sequencing technologies has been extended to metagenomics, that is to DNA extracted directly from an environmental sample. Raw samples of microbial organisms can be easily sequenced in parallel and this new culture-independent practice allows for unanimous study of all genomes recovered from an environmental community. This opportunity opens doors to identifying known and novel organisms in a microbial community and draws attention to understanding their genetic diversity and vast network of interactions on a global scale. One of the greatest challenges of modern-day sequence analysis is to traceback this massive amount of sparse (read) data onto the genomes of original or closely related organisms.

Thus, for seeking out divergent species in metagenomics studies or mapping highly erroneous reads generated by new technologies such as PacBio, more sensitive tools are required. For this task, mapping tools such as BLAST [Altschul et al., 1990] and SHRiMP2 [David et al., 2011] would be more appropriate as they are capable of handling more errors accumulated through biological evolution, as well as those introduced by HTS technologies. An additional feature in BLAST, is the evaluation of an alignment's biological integrity using the expectation value (E-value), which allows to narrow down the number of false alignments generated by pure chance alone. The major drawback of the aforementioned tools is their slow execution speed, especially for sequences with lengths longer than 500 bp. Moreover, even the extremely short (7 bp) and inexact seeds have difficulties capturing all polymorphism errors, especially in 454 and PacBio data, without jeopardizing the accuracy of an alignment.

4.2 SortMeDNA

4.2.1 Principle of the algorithm

The inputs to SortMeDNA are a collection of reads and a reference database composed of genomic sequences onto which the reads should be mapped. The algorithm has three principal steps (also illustrated in Figure 4.1),

1. a *primary seed-search filter*, during which we search for seed matches between the read and the reference database,
2. a *secondary seed-cluster filter*, where these short seeds are aggregated into longer matches using a longest increasing subsequence (LIS),
3. *alignment followed by selection*: we refine the previously found (LIS) clusters by performing Smith-Waterman local alignment beginning near the LIS, and finally select the read according to the statistical significance of its alignment.

Steps 1-3 are organized in a cyclic manner. In Step 1, SortMeDNA does not search for seeds at every position of the read but shifts a window of length k across the read every x characters for pass 1, then every y characters for pass 2 and lastly every z characters for pass 3 as depicted in Figure 4.1, where $x > y > z$ in order to granulate the search (by default $x = k$, $y = \frac{k}{2}$ and $z = 3$). The search for seeds only advances to subsequent passes if (a) the threshold number of seeds did not match to the database (by default 2), (b) the matching seeds did not form a long enough LIS (by default 2, same threshold value as required to pass the primary seed-search filter), or (c) the LIS did not extend to a significant enough alignment (by default E-value=1). In every case, the algorithm will backtrack through the filters in Figure 4.1 to find new LIS candidates or collect more seeds using a finer granularity search. In the following of this section, we describe each of these three steps in further detail.

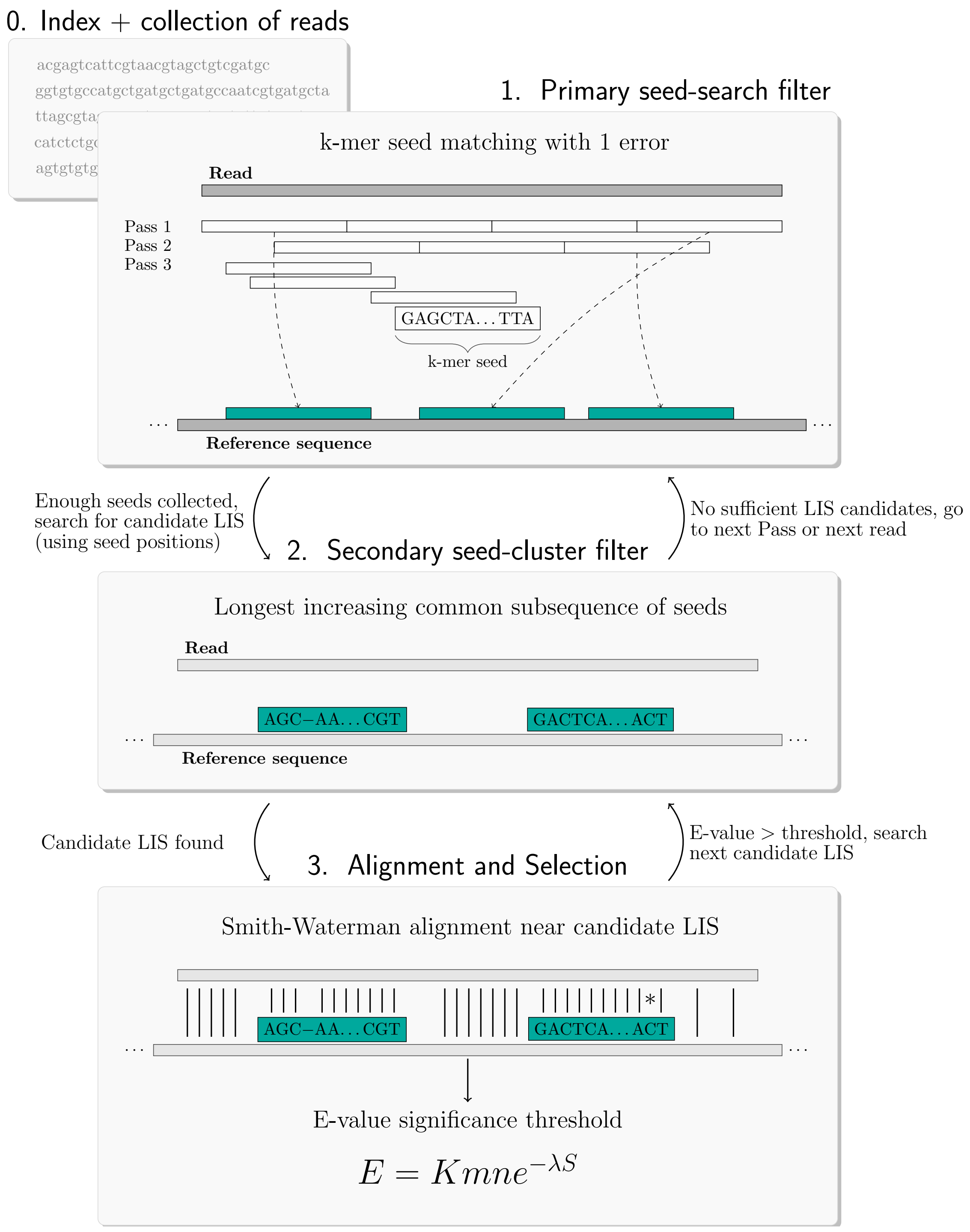
4.2.1.1 Primary seed-search filter

The primary filter uses the approximate seeds of length k with one error, together with the mini-burst tries presented in Chapter 2. We traverse the entire mini-burst trie collecting all matches and stop only if we find an exact match or we reach the end of the mini-burst trie. This heuristic to stop traversal after an exact match has been found is based on the assumption that 0-error hits are more significant than 1-error hits. This heuristic can be turned off to search for **both** *exact* **and** *all 1-error* matches, which increases sensitivity very slightly (often by less than 1%), since the rare cases where 1-error hits are more significant than 0-error hits are examined, but decreases the speed by up to four-fold in practice. Since the error can occur in the first or second part of the k-mer, we apply the same search principle on the reverse k-mer.

4.2.1.2 Secondary seed-cluster filter

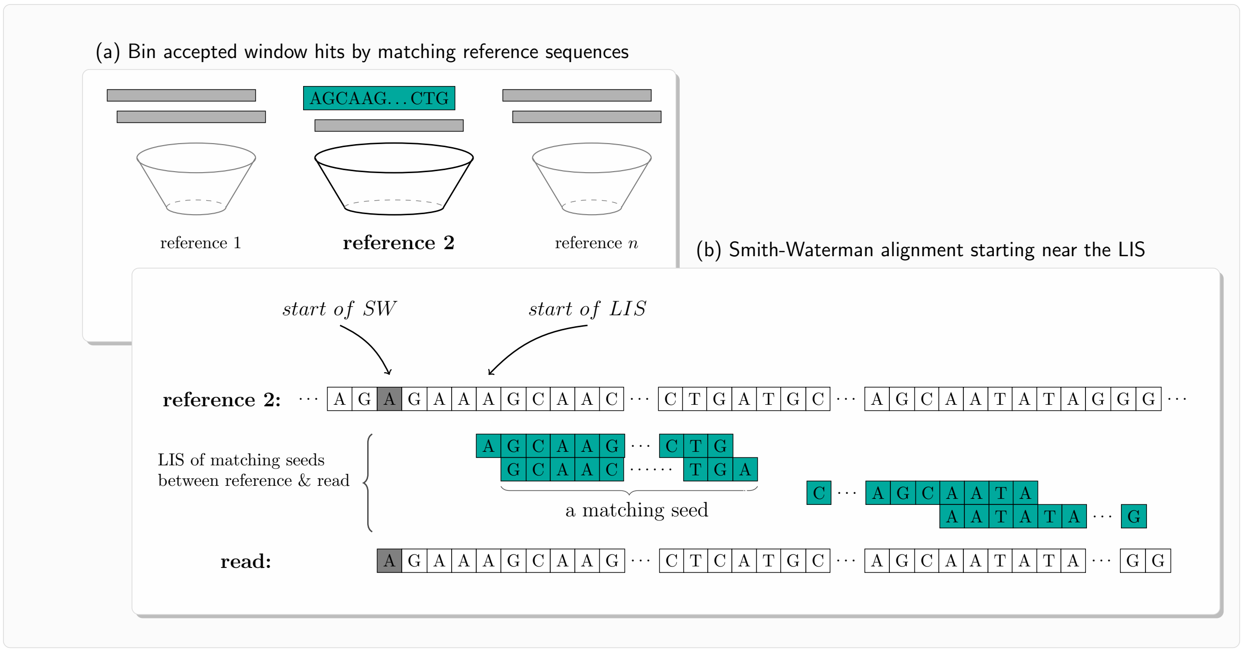
In this step, we use the collection of matching seeds accumulated during the primary seed-search filter step to isolate longer homologous regions between the read and a reference sequence. We

Figure 4.1: SortMeDNA algorithm pipeline


0. Index + collection of reads
acgagtcattcgtaacgtagctgtcgatgc
ggtgtgccatgctgatgctgatgccaatcgtgatgcta
1. Primary seed-search filter
k-mer seed matching with 1 error
Read
Pass 1
Pass 2
Pass 3
GAGCTA...TTA
k-mer seed
Reference sequence
Enough seeds collected, search for candidate LIS (using seed positions)
No sufficient LIS candidates, go to next Pass or next read
2. Secondary seed-cluster filter
Longest increasing common subsequence of seeds
Read
AGC–AA...CGT
GACTCA...ACT
Reference sequence
Candidate LIS found
E-value > threshold, search next candidate LIS
3. Alignment and Selection
Smith-Waterman alignment near candidate LIS
AGC–AA...CGT
GACTCA...ACT
E-value significance threshold
$E = Kmne^{-\lambda S}$

begin by binning the seeds to their corresponding reference sequences, as shown in Figure 4.2(a), by using the $(k+1)$-mer list of positions to link each matching seed to the original location on a reference sequence. Next, we isolate regions of length equal to the read on each reference sequence containing a threshold number of matching seeds (by default 2). Lastly, we compute a variant of the longest increasing subsequence (LIS) on the seeds' positions on the read relative to a region on the reference sequence, as shown in Figure 4.2(b). If the LIS is composed of a threshold number of seeds (by default 2), we proceed to the final step of alignment and selection.

Figure 4.2: Longest increasing subsequence and local alignment


(a) Bin accepted window hits by matching reference sequences
AGCAAG...CTG
reference 1
reference 2
reference n
(b) Smith-Waterman alignment starting near the LIS
start of SW
start of LIS
reference 2:
LIS of matching seeds between reference & read
a matching seed
read:

4.2.1.3 Alignment and final selection

The final step of the algorithm is to extend the LIS obtained with the secondary seed-cluster filter into a longer sequence alignment and evaluate the biological significance of its score using the E-value. Regardless of where the LIS is positioned between the read and the reference sequence, local alignment begins at the head of the read and is carried throughout the length of the read. In boundary conditions, if the read overhangs the reference sequence, the starting position and length of alignment are modified accordingly. The sequence alignment is performed using the SSW library [Zhao et al., 2012], which implements the Smith-Waterman local alignment algorithm using SIMD instructions. Once an alignment is made, we evaluate the significance of its score by computing the E-value. Using the ALP program [Sheetlin et al., 2005], SortMeDNA computes the Gumbel parameters (see Section 1.2.4) on-the-fly subject to a user-configurable alignment scoring scheme (match, mismatch, gap open and gap extend scores) and the relative background frequencies of the nucleotides in the database. We note here that for any given set of Gumbel parameters and E-value, we can compute the minimum Smith-Waterman score required to achieve the same E-value using Equation 4.1,

$$minScore = \frac{\ln(Kmn) - \ln(E)}{\lambda} \tag{4.1}$$

We will use this *minScore* in the following sections to help evaluate performances of tools which do not apply the E-value with respect to SortMeDNA.

4.2.2 Predefined parameter setting

SortMeDNA comes with two preset modes which must be specified during index construction: `-fast`, recommended for ∼99% sequence similarity, and `-sensitive`, recommended to ∼75-99% sequence similarity. These two modes are governed by the same algorithm and simply correspond to two predefined parameter settings covering all main application fields.

Length of the seed: SortMeDNA uses seeds of size $k \in [8, 26]$. By default, the length of the seed is $k = 24$ nucleotides for fast mapping and $k = 18$ for sensitive mapping. Lengths $k = 24$ and $k = 18$ were chosen for their best performance tradeoff between sensitivity and accuracy based upon tests for lengths $[14, 26]$, although the user may vary k to obtain more sensitive ($k < 18$) or faster ($k > 24$) results. Therefore, in the fast mode the interval sizes between seeds are set to 24, 12 and 3 nt across the read, whereas for the sensitive mode they are set to 18, 9 and 3 nt.

4.2.3 Implementation and Availability

SortMeDNA is written in C/C++ and freely distributed under the GNU general public license. The input is a set of letter space reads produced by second or third-generation technologies (tested on Illumina, 454, Ion Torrent and PacBio), and a multi-FASTA file of the reference sequences. SortMeDNA uses the ALP program to calculate the statistical parameters for the E-value, the SSW library to perform rapid Smith-Waterman local alignment using SIMD instructions, and the CMPH library for minimal perfect hashing of the k-mer position tables. It also provides support for multi parallelism during the filtering and mapping steps using OpenMP. The standalone version is distributed as a complete software package along with all the necessary libraries, and can be easily installed using Autotools. Additionally, SortMeDNA is available as a Galaxy wrapper.

The inputs to SortMeDNA are a collection of reads, in multi-FASTA or FASTQ format, and a reference database composed of genomic sequences, in multi-FASTA format, onto which the reads should be mapped. SortMeDNA can be used as a filter, to generate only a multi-FASTA or FASTQ file of reads matching to the reference database with an E-value lower than a given threshold, or as an aligner to generate full alignments in the SAM or a visual BLAST-like format.

4.3 Performance results

In this section, we test the performance of SortMeDNA against a representative selection of read mappers: Bowtie2, BWA-SW, SHRiMP2. We also use BLASTN as a control of the mapped data or on small datasets. Our tests take into consideration sequencing data generated by Illumina, 454, Ion Torrent and PacBio technologies, varying in lengths between 75-10,000 nt and encapsulating both substitution and indel errors. They also deal with a variety of applications from genomic and metagenomic projects.

Datasets: Firstly, we consider classical genomic applications such as genome resequencing and aligning data to very low-divergent genomes (∼99% similarity based on whole-genome phylogeny). We use simulated reads from the *A. thaliana* genome (Test 1), and Illumina and 454 reads from the human genome (Test 2). In both tests, we demonstrate the performance of SortMeDNA with respect to Bowtie2 and BWA-SW, both tools optimized for such applications, as well as SHRiMP2, which is a more lenient tool for mismatch errors and optimized for short reads.

Secondly, we measure the software's efficacy for capturing sequence diversity by mapping data against variant strains, such as those of *S. aureus* in Test 3 and *A. thaliana* in Test 4, and to closely related species (75-99% similarity), such as a bacterial metagenome of the human intestine in Test 5 and that of the arctic soil in Test 6. Lastly, in Test 7 we use PacBio reads, which are longer and exhibit higher error rates, to align strains of *V. cholerae*.

Parameters: For all tests, we apply uniform parameter settings. For genomic applications presented in Test 1 and Test 2, Bowtie2, SHRiMP2 and BWA-SW were run using default parameters, which are well-suited for this type of application. SortMeDNA was run using the `-fast` option. For all other tests (Tests 4-7), Bowtie2, SHRiMP2 and BWA-SW were run using two modes: default parameters, and more sensitive parameters that were determined manually in Test 3 based on sensitivity and selectivity performances for mapping *Staphylococcus* strains. The more sensitive parameters apply the same Smith-Waterman scoring scheme (match=2, mismatch=-3, gap open=-5 and gap extend=-2) which is currently the default for BLASTN. In addition, `-very-sensitive-local` was used for Bowtie2, `-full-threshold 50` (as a raw score) for SHRiMP2 and `-z 100` for BWA-SW. SortMeDNA was run with the `-sensitive` option, as it is tailored specifically for these kinds of applications.

Computer architecture: All tests were performed using one core of an Intel(R) Xeon(R) E5606 @ 2.13GHz processor.

4.3.1 Test 1: Mapping and alignment accuracy using *Arabidopsis thaliana* genome

In this experiment, we evaluate accuracy and running time of all software for simulated reads for a variety of sequencing technologies. We simulated 5,000,000 Illumina and 2,000,000 Roche 454 using Mason [Holtgrewe, 2010] and Ion Torrent reads using DWGSIM [Li, 2012] from the latest *A. Thaliana* assembly TAIR10. In order to obtain reads with a unique best-matching position, the program RepeatMasker [Smit et al., 2008] was used to mask repeat sequences (downloaded from the Plant Repeat Databases [Ouyang and Buell, 2004]) on the TAIR10 chromosomes and their known original positions. This alignment evaluation was performed using the benchmarking method Rabema [Holtgrewe et al., 2011] for Illumina and 454 reads, and the dwgsim_eval program from the DWGSIM package for Ion Torrent reads.

As shown in Table 4.1, SortMeDNA mapped 100% of the reads with minimum 98.38% of the mappings correctly associated to the original position on the genome, which makes it one of the most sensitive and accurate tools. Given that SortMeDNA computes alignments for all high-scoring candidate positions before selecting the best one, similar to SHRiMP2, it retains a very reasonable speed being the fastest tool in this example for 454 and Ion Torrent reads.

4.3.2 Test 2: Genome resequencing of *Homo sapiens* using Illumina and 454 data

In this classical experiment, we mapped data from two different human genomes against the latest build GRCh37 (masked and unmasked versions) using the tools SortMeDNA, Bowtie2, BWA-SW and SHRiMP2 in their default modes. The first data set was for a Mongolian individual sequenced using the Illumina Genome Analyzer II, and the second data set was for James Watson sequenced using the 454 GS FLX platform. The results are presented in Figure 4.3.

SortMeDNA found the second highest number of reads for the Illumina reads (after Bowtie2)

Table 4.1: Rabema benchmark for 5,000,000 Illumina reads (100 nt) and 2,000,000 454 reads (∼250 nt), dwgsim_eval for 2,000,000 Ion Torrent reads (150 nt) from masked *A. thaliana*.

sequencing technology	software	mapped reads [%] (of total)	correctly mapped reads [%] (of mapped)	time [hr:min]
Illumina	SortMeDNA `(-fast)`	100	98.38	0:30
	Bowtie2	99.99	98.38	0:30
	SHRiMP2	99.99	98.35	2:20
	BWA-SW	99.88	98.35	0:46
454	SortMeDNA `(-fast)`	100	99.04	0:28
	Bowtie2	100	99.01	1:30
	SHRiMP2	99.99	99.04	8:10
	BWA-SW	99.99	99.00	0:34
Ion Torrent	SortMeDNA `(-fast)`	100	98.78	0:16
	Bowtie2	99.99	98.78	0:37
	SHRiMP2	99.99	98.78	2:10
	BWA-SW	99.60	98.00	0:27

and the highest number of reads for the 454 reads and in time comparable to Bowtie2. To investigate the significance of the reads found only by one tool, we will use Equation 4.1 to compute the minimal threshold score $minScore_{evalue}$ required to reach E-value=1 (default for SortMeDNA) for each tool (the Gumbel parameters were computed using the program ALP [Sheetlin et al., 2005]). This computed $minScore_{evalue}$ is given in column four of Tables 4.2 and 4.3. By construction, all reads found by SortMeDNA are guaranteed to be above the score threshold for E-value=1, which is not the case for the other tools. For example, the majority of reads found only by Bowtie2 do not surpass the $minScore_{evalue} = 62$. As a consequence, SortMeDNA finds strictly more reads with a statistical significant alignment than most other tools, expressed either in percentage or in absolute terms (see column 5 of Table 4.2 and 4.3). SHRiMP2 uses a percent cutoff threshold in the length of the read rather than the raw score (default set to 50%). Thus, the minimal score of all 31,771 Illumina reads found only by SHRiMP2 is 505 (exactly half of 10×101) which is also (as for SortMeDNA) well above their $minScore_{evalue} = 347$.

In terms of time, the FM-index based tools (BWA-SW and Bowtie2) are the fastest tools for this type of application since they search for exact seeds and output an alignment based on one or two best matching hits. Although BWA-SW implicitly allows mismatch and indel errors in its seeds, the default Z-best score used to accelerate the search looks only at the first best matching seed and therefore can miss many other candidate matches. To elaborate on this point, we can compare the alignment scores of the reads found by two or more tools by running them with the same set of Smith-Waterman parameters. This was done for SortMeDNA and Bowtie2 using the current default Smith-Waterman parameters for BLASTN (match=2,mismatch=-3,gap open=-5, gap extend=-2). For the Illumina reads, together both tools found 8,909,532 reads. Of these 8,909,532 reads, 1,045,582 did not agree on the alignment position by at least 5 nucleotides. For 701,359 of 8,909,532 reads (approximately 8%), SortMeDNA found better alignments. For 89,674 of 8,909,532 reads (approximately 1%), Bowtie2 found better alignments. Together, both tools found 254,549 alignments which vary at the alignment position but have the same alignment score. For the 454 reads, together both tools found 4,651,197 reads. Of these 4,651,197 reads, 321,173 did not agree on the alignment position by at least 5 nucleotides. For 268,194 of 4,651,197 reads (approximately 5%), SortMeDNA found better alignments. For 23,737 of 4,651,197 reads (approximately 0.5%), Bowtie2 found better alignments. Together, both tools found 29,242 alignments which vary at the alignment position but have the same alignment score.

SortMeDNA can also be run with the `-feeling-lucky` parameter which will force the program to stop searching for better alignments once an alignment reaching the threshold E-value has been found. This parameter accelerates the program significantly (as indicated with the * time in Figure 4.3), although now for the reads which do not match at the alignment position and score, SortMeDNA will give better alignments than Bowtie2 for ∼6% of the reads and Bowtie2 will give better alignments than SortMeDNA for ∼8% of the reads. Therefore the quality of SortMeDNA's alignments become similar to those of Bowtie2 with the option `-feeling-lucky`, however they are still always guaranteed to reach the E-value threshold.

Figure 4.3: The distribution of Illumina and 454 reads mapped against masked and unmasked versions of GRCh37 for each tool and compiled into Venn diagrams. For Illumina: 10,000,000 reads (101 nt) from a Mongolian individual's genome. For 454: 4,733,349 reads (∼265 nt) from James Watson's genome. For each software, we report the number of reads mapped and the computation time ([hr:min]). For SortMeDNA, the time represented by * is obtained using the option `-feeling-lucky` which stops the algorithm from searching for better alignments once the first alignment reaching the E-value threshold is found.

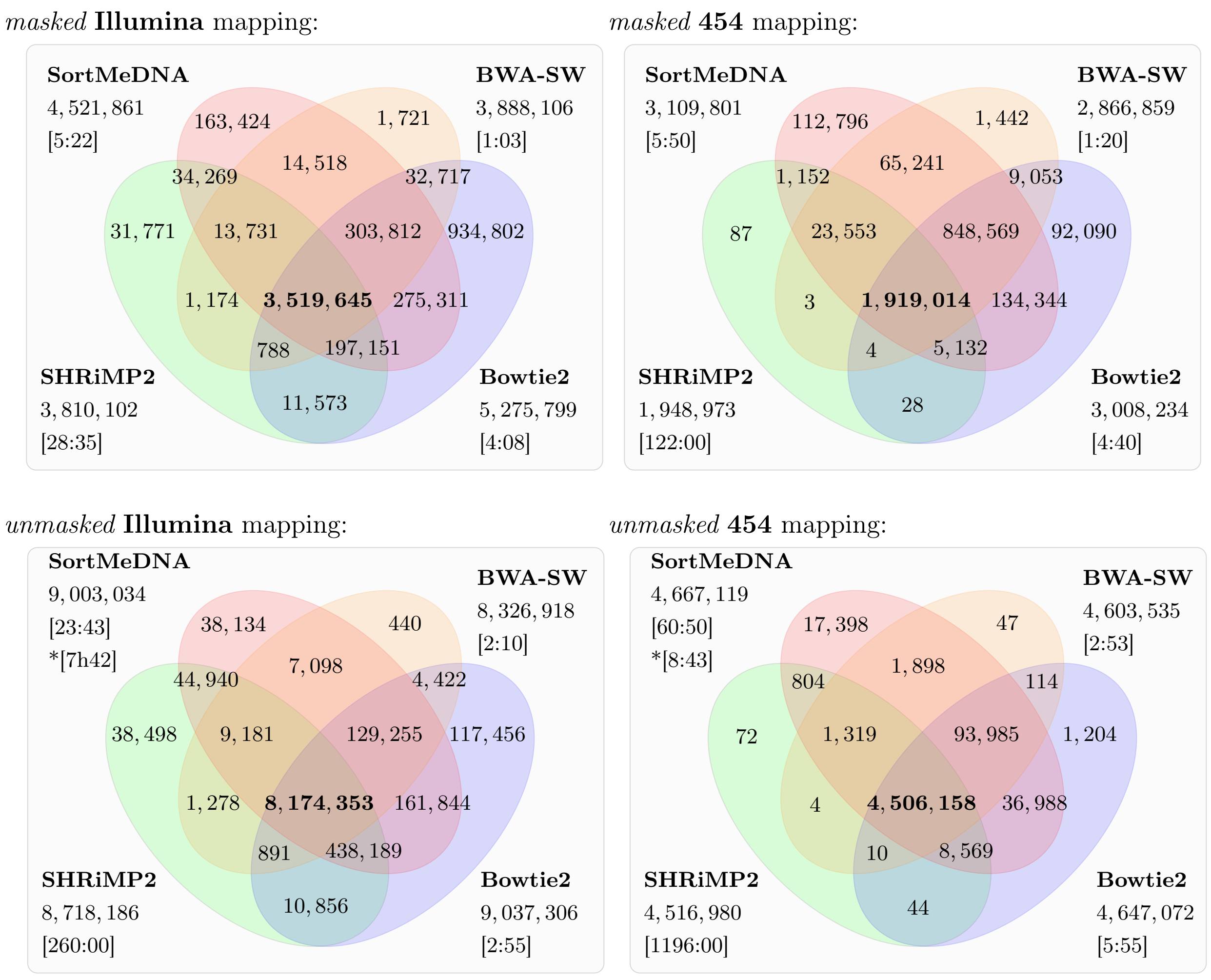


Table 4.2: Analysis of reads found by only one read mapper in default mode for *masked* GRCh37 (see Venn diagrams in Figure 4.3). Equation 4.1 was used to compute the $minScore_{evalue}$ for E-value=1 for each software using their default Smith-Waterman parameters.

sequencing technology	software	# reads found by only one software	$minScore_{evalue}$	# reads with $\geq minScore_{evalue}$	% reads with $\geq minScore_{evalue}$
Illumina	SortMeDNA	163,424	69	163,424	100.0
	SHRiMP2	31,771	347	31,771	100.0
	Bowtie2	934,802	62	49,557	5.3
	BWA-SW	1,721	31	1,538	89.3
454	SortMeDNA	112,796	69	112,796	100.0
	SHRiMP2	87	348	85	97.7
	Bowtie2	92,090	63	14,426	15.6
	BWA-SW	1,442	31	1,304	90.4

Table 4.3: Analysis of reads found by only one read mapper in default mode for *unmasked* GRCh37 (see Venn diagrams in Figure 4.3). Equation 4.1 was used to compute the $minScore_{evalue}$ for E-value=1 for each software using their default Smith-Waterman parameters.

sequencing technology	software	# reads found by only one software	$minScore_{evalue}$	# reads with $\geq minScore_{evalue}$	% reads with $\geq minScore_{evalue}$
Illumina	SortMeDNA	38,134	68	38,134	100.0
	SHRiMP2	38,498	345	38,498	100.0
	Bowtie2	117,456	62	7,596	6.4
	BWA-SW	440	31	400	90.9
454	SortMeDNA	17,398	69	17,398	100.0
	SHRiMP2	72	347	72	100.0
	Bowtie2	1,204	63	134	11.1
	BWA-SW	47	31	13	27.6

4.3.3 Test 3: Sensitivity and selectivity using *Staphylococcus* strains

The purpose of this test was to measure the sensitivity and selectivity of all tools when mapping reads from a close strain to a reference genome using default and varying parameters. We define *sensitivity* as the number of reads mapped from *S. aureus 71193* to the reference genome *S. aureus T0131* (97% whole-genome similarity), and *selectivity* as the number of random reads mapped to the *S. aureus T0131* genome. For sensitivity, we simulated 3 sets of reads: 5,000,000 reads for Illumina and 2,000,000 reads for 454 technologies using Mason, and 2,000,000 reads for Ion Torrent technology using DWGSIM. For selectivity, we randomly generated 10,000,000 reads of length 100 nt (for Illumina), 10,000,000 reads of length 150 nt (for 454), and 10,000,000 reads of length 250 nt (for Ion Torrent) using nucleotide background frequencies of *S. aureus T0131*.

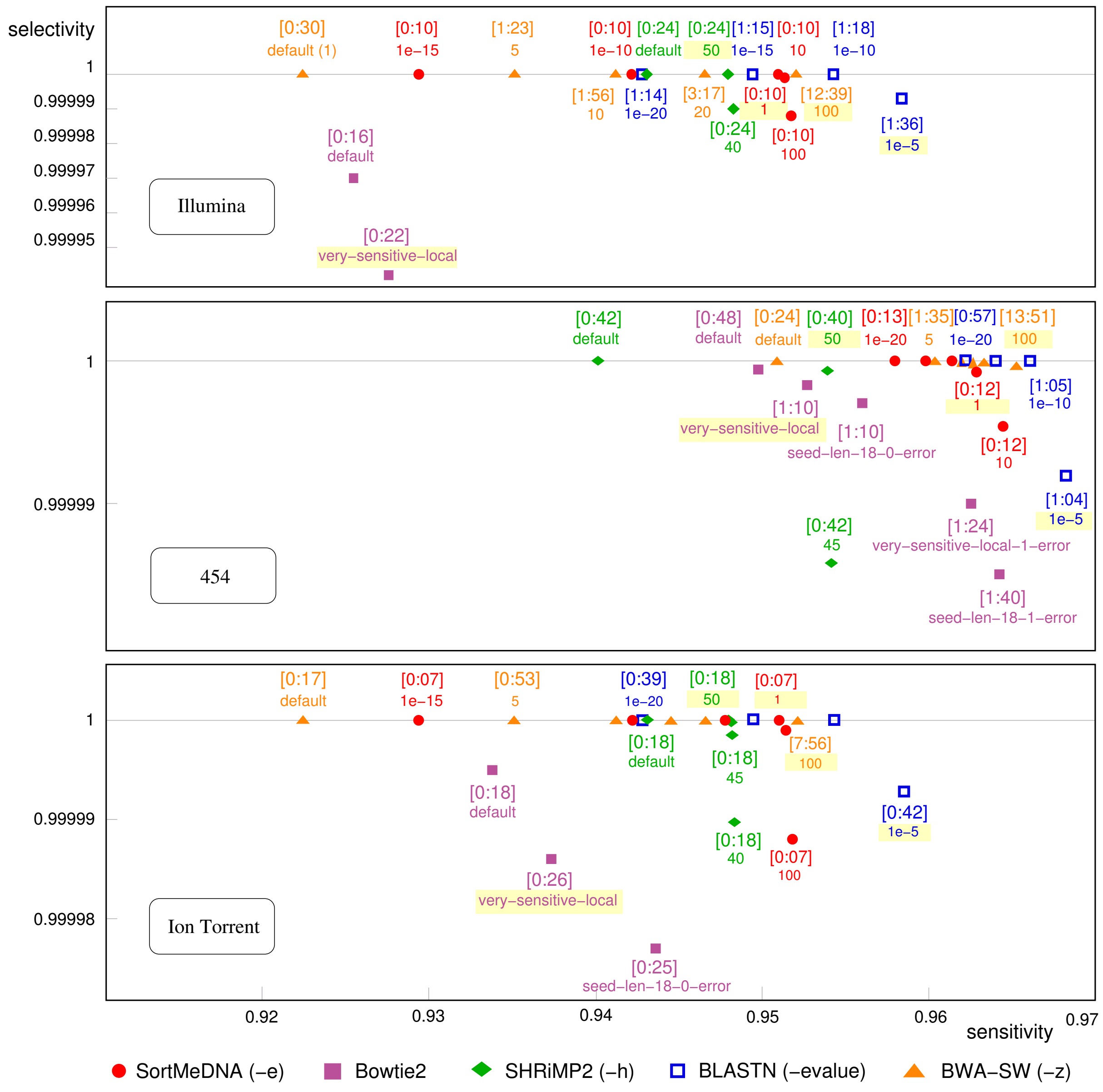
For SHRiMP2, Bowtie2 and BWA-SW, we applied variations to the parameters that most influenced the sensitivity of the results, being the E-value for SortMeDNA and BLASTN, the Smith-Waterman threshold cutoff score for SHRiMP2 and the Z-best score for BWA-SW. In default mode, SHRiMP2 makes global alignments and uses the Smith-Waterman threshold cutoff as a percentage of the maximum score rather than as a raw score, however since local alignments are not required to span the length of the entire read, we prefer varying the threshold cutoff score as a raw score. Bowtie2 was run in the default mode (seed length = 20, 0-error in seed, 8-12 nt interval size), the `-very-sensitive-local` preset mode (seed length = 20, 0-error in seed, 6-8 nt interval size), the `-very-sensitive-local` preset mode with 1 mismatch in the seed, and two more sensitive settings using the same seed length as SortMeDNA (seed length = 18, 0-error in seed, 6-8 nt interval size) and (seed length = 18, 1-error in seed, 6-8 nt interval size). To maintain regularity between software for varying parameters, the Smith-Waterman alignment parameters were set to the BLASTN default values (reward = 2, penalty = -3, gap open = -5, gap extend = -2) for all tools. The results of this test are shown in Figure 4.4. The best-performing varied parameters were subsequently chosen to be the sensitive parameters for Tests 4-7 and are highlighted in yellow.

SortMeDNA performs very well across all read types, showing minor variance in the speed and accuracy among different E-values. The default mode for SHRiMP2 always offers the highest selectivity than its varied modes, but a better performance trade off can be achieved by using a sensitive setting with threshold score cutoff set to 50, gaining a slight 1% in sensitivity without loss in selectivity. As with SortMeDNA, the runtime for SHRiMP2 is not strongly affected by varying the parameters. For BLASTN, an E-value of $\leq 1e$-5 gives the best results, although the higher the E-value for BLASTN, the longer the runtime of the program, which can increase mani-fold from E-value of $1e$-20 to 10. Bowtie2 shows the poorest selectivity for all settings compared to other tools, and for the setting (seed length = 18, 1-error, 6-8 nt interval size) it has a similar sensitivity to SortMeDNA but at a five-fold decrease in speed. The default setting for Bowtie2 work well for genome resequencing but not are sensitive enough for low-divergent ($<$ 98%) sequences. BWA-SW is the least sensitive tool, but with the Z-best score set to 100, it becomes one of the most sensitive tools while maintaining a very high selectivity. However, increasing the Z-best score also increases the speed from a few minutes to a few hours, since the algorithm must search for more matching seeds in the index.

4.3.4 Test 4: Swedish environmental study of *A. thaliana* using Illumina data

In test 1, we have considered simulated reads from *A. thaliana* genome. In this experiment, we test the performance of SortMeDNA, Bowtie2, BWA-SW and SHRiMP2 for capturing ge-

Figure 4.4: Performance results for SortMeDNA, SHRiMP2, BLASTN, Bowtie2 and BWA-SW for mapping *S. aureus 71193* reads against *S. aureus T0131* reference genome

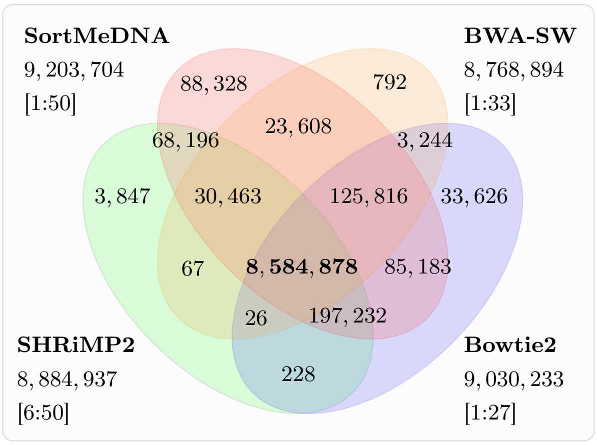


netic intraspecies diversity using sequenced strains of *A. thaliana* collected across Sweden (study SRP012869). After filtering low-quality paired reads from accession SRR519675, we took the first 10,000,000 forward reads and mapped them against the latest (masked) *A. thaliana* build TAIR10 (the same reference database used in Test 1).

As shown in Figure 4.5, SortMeDNA was able to capture more reads (92.03%) than all other tools and at a speed similar to Bowtie2 and BWA-SW. In default mode, BWA-SW shows the lowest sensitivity of 87.69%, which can be increased by 3% in sensitive mode (Z-best score 100) but with a nearly 100x drop in speed. In the sensitive mode, all programs mapped more reads with the least number of mapped reads, 90.59%, made by SHRiMP2, closely followed by Bowtie2 at 90.61%.

Figure 4.5: 10,000,000 Illumina reads (100 nt) from multiple strains against masked *A. thaliana* genome

in **default** mode:



in **sensitive** mode:

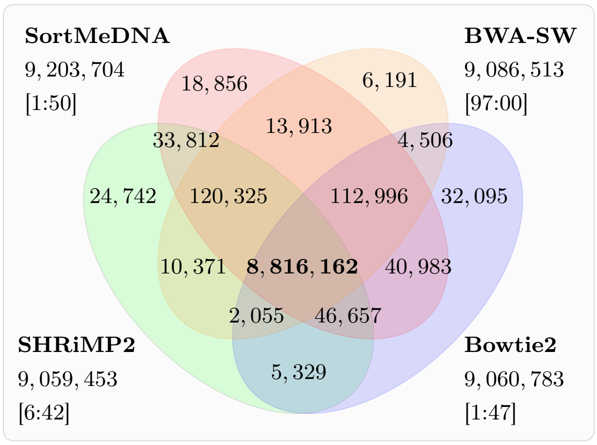


Table 4.4: Analysis of reads found by only one read mapper in sensitive mode for *A. thaliana* (see Venn diagram in Figure 4.5) according to BLASTN. The table shows how many reads found by only one of the tools are also found by BLASTN having alignment length ≥ 30, identity ≥ 70% and E-value ∈ $[1, 0.01, 1e\text{-}5, 1e\text{-}10]$.

E-value	SortMeDNA (18,856)		SHRiMP2 (24,742)		Bowtie2 (32,095)		BWA-SW (6,191)	
	# reads	% reads	# reads	% reads	# reads	% reads	# reads	% reads
1	18,606	98.6	22,328	90.2	21,447	66.8	5,844	94.3
0.01	18,571	98.4	21,574	87.1	21,447	66.8	5,722	92.4
1*e*-5	17,768	94.2	19,627	79.3	14,994	46.7	5,534	89.3
1*e*-10	10,533	55.8	9,762	39.4	391	1.2	1,854	29.9

Further analysis was made to examine the reads found only by any one tool in sensitive mode. Firstly, it appears that SortMeDNA mapped more reads with indel errors than all other tools. About 63% of the 18,856 reads found only by SortMeDNA have indel errors, whereas for SHRiMP2, Bowtie2 and BWA-SW, these values are 45%, 13% and 32%, respectively. Secondly, we classified those reads according to BLASTN based on alignment length ≥ 30, identity ≥ 70% and E-value threshold ∈ $[1, 0.01, 1e\text{-}5, 1e\text{-}10]$, as shown in Table 4.4. At E = 1*e*-10, over 55% of reads found only by SortMeDNA remain significant, the highest of all tools, whereas only about 1% for Bowtie2. Since SortMeDNA applies statistical computations to classify reads, most of the resulting matches retain higher scores than what would be expected in a random alignment. For SortMeDNA, an E-value = 1 is comparable to having a minimal Smith-Waterman alignment score of 61 for this Illumina dataset, meaning any alignments with scores below this value can possibly occur by chance alone. SHRiMP2, BWA-SW and Bowtie2, all use a minimal score threshold to validate a Smith-Waterman alignment, which is set to 50, 60 (varies with length) and ∼56 (varies with length) respectively for each tool with the default BLASTN Smith-Waterman parameters. BWA-SW applies the closest matching minimal cutoff score (60) to the one computed by SortMeDNA for E-value = 1, and it is also the only other tool that has almost 90% of its possible false-positive reads (6,191) reaching high significance at E = 1*e*-5 according to BLASTN.

4.3.5 Test 5: HMP mock community samples

Metagenomic studies are important for identifying known and novel organisms in a microbial community and understanding their genetic diversity. The mock metagenomic community from the Human Microbiome Project (HMP) is a controlled study of 21 known organisms commonly found in the human body, which have been sequenced using the Illumina and 454 technologies [Turnbaugh et al., 2007]. Although the original positions of the sequenced reads are not available, this study provides a true metagenomic sample with all of the constituting species known.

The main approach to classifying metagenomic reads is to align them against a database of known reference sequences. Aligning these reads back onto the original genomes is a relatively straightforward task, since many of the species are sufficiently diverged and the only errors in the set are those introduced by the sequencing technologies. When run in default mode, each tool mapped relatively equal abundance of each species as shown in Figures 4.6-4.7. The only exception is SHRiMP2 for 454 reads, that found marginally less reads than all other software and experiences a large spike in execution time, likely due to the higher indel rate and the long and varying read lengths (49-2048 nt).

In real-life projects, fully sequenced genomes or complete sequences exist only for a small fraction of the millions of estimated microorganisms on earth [Kallmeyer et al., 2012], and often the reference database is made up of closely related species or strains. Real-world environmental samples contain thousands of unknown organisms including those for which reference genomes are not available. To simulate this scenario, we designed a second test based on the HMP community for assessing the capability of read mapping tools to identify reads from closely related species. For that purpose, as shown in Table 4.5, we selected substitute species for 14 of the 21 original organisms. The substitute species were chosen with divergence beginning at the genus level having 75-99% whole-genome similarity to the original genomes. We selected substitute species based on whole-genome phylogeny rather than the traditional 16S rRNA marker gene, as they can give higher resolution to species diversity at short evolutionary time scales [Segata and Huttenhower, 2011]. The whole-genome similarity measures were estimated based on the distance matrix generated by CVTree [Qi et al., 2004], a phylogenetic tree reconstruction tool which computes distances using compositional vectors calculated based on frequency of amino acids strings of whole genomes. In Table 4.5, entries marked with a '$\star$' define organisms for which substitution species were given and entries without a marked '$\star$' were kept in the set as control sequences. Figure 4.8 shows a phylogenetic tree between the original species in the HMP mock community and the substitute species. Figure 4.9 shows that the evolutionary distances for the original species within the HMP community are much greater ($\sim$0.42) than the distances between the substitute species **and** the species for which substitutes were given for ($\sim$0.2). This is simply a confirmation that the reads should map to the substitute species rather than to other original species in the set.

When mapping reads onto their substitute species, the results show more fluctuation amongst all tools as illustrated in the mirror image of the original species in Figures 4.6-4.7. Although Bowtie2 displays the fastest execution time, for both Illumina and 454 reads it found significantly fewer reads for most substituted species than all other tools. For Illumina reads, SortMeDNA delivers comparable results to both SHRiMP2 and BLASTN, and together with SHRiMP2, nearly 13x faster than BLASTN. SHRiMP2 was able to identify more Illumina reads than SortMeDNA possibly due to the fact that bacterial genomes contain many coding regions, up to 66% GC-content for the substitute species in Test 2 (ex. *Deinococcus geothermalis*). Two of the 3 default

Figure 4.6: Performance results for SortMeDNA, SHRiMP2, BLASTN, Bowtie2 and BWA-SW for mapping HMP Mock Community **Illumina** reads against original and substitute species

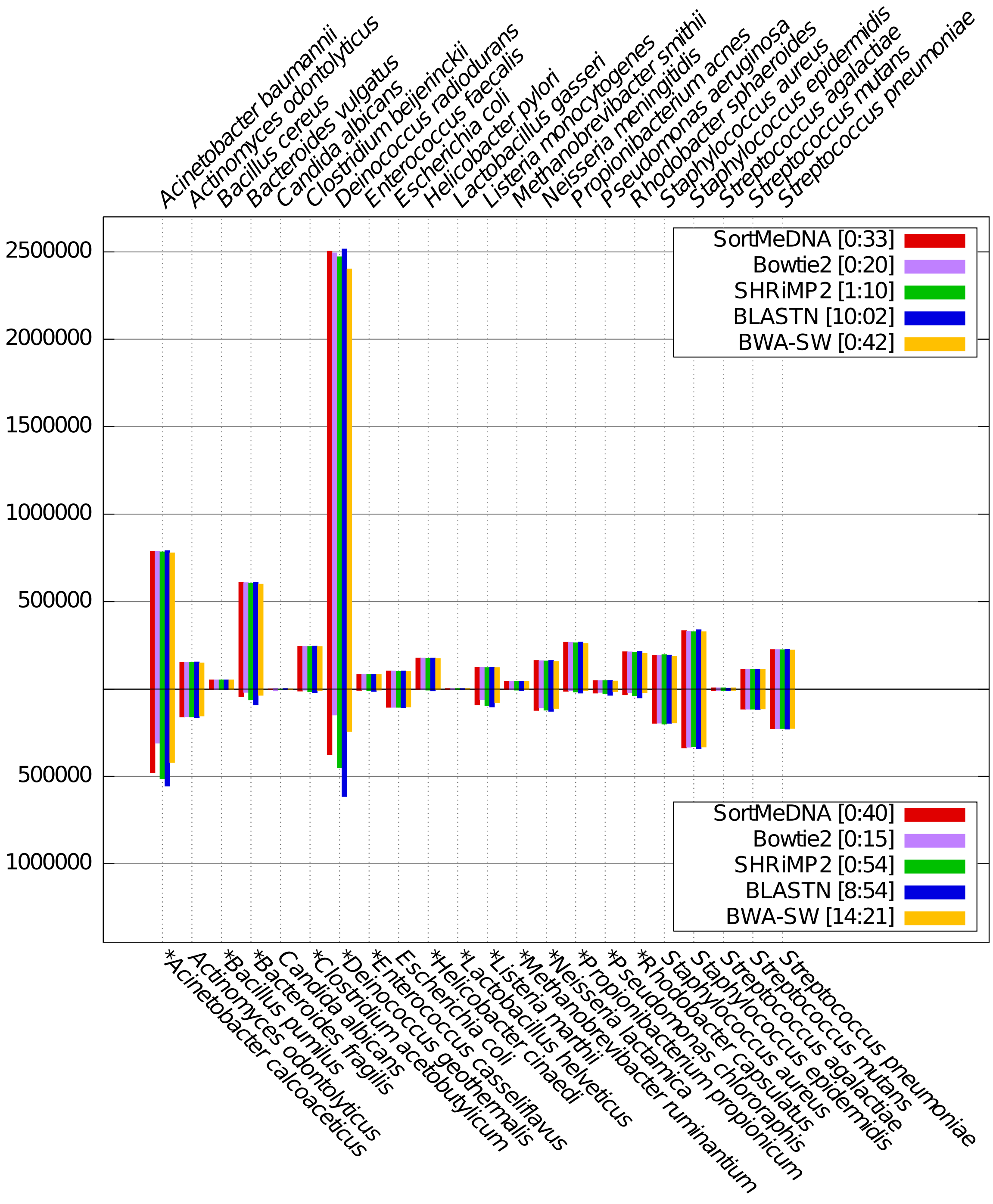
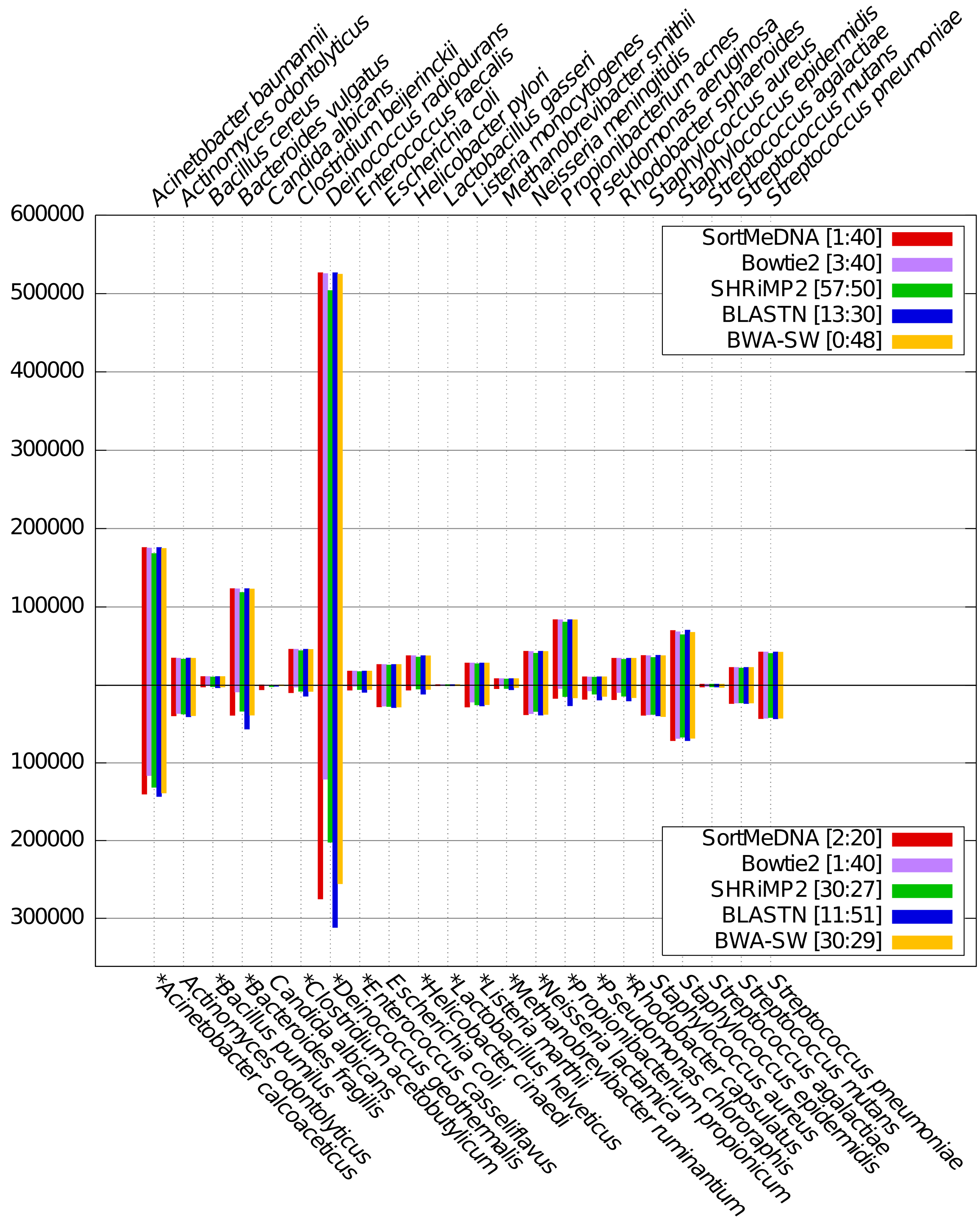


Figure 4.7: Performance results for SortMeDNA, SHRiMP2, BLASTN, Bowtie2 and BWA-SW for mapping HMP Mock Community **454** reads against original and substitute species



seeds used by SHRiMP2 allow up to 7 mismatches, which may give an advantage over a seed allowing one error in an arbitrary position for locating coding regions, since nucleotides coding for the same amino acid often variate at the third nucleotide. For 454 reads, SortMeDNA remains robust to the long and varying read lengths, and provides results comparable to BLASTN at 6x the speed.

4.3.6 Test 6: Metagenomic study of arctic soil using Ion Torrent data

Sequencing technologies generating long reads substantially reduce the challenges faced by short-read comparative analyses, since more biological information is covered by each read. In metagenomics, this property translates into better detection of distant homologs and overall classification of organisms [Wommack et al., 2008]. In this experiment, we test the mapping performance of all tools using an arctic soil metagenome sequenced using Ion Torrent technology. In total, there are 610,536 reads of varying lengths (18-973 nt) and average length of 210 nt (accession SRR678264). The reference database was constructed using 12 bacteria commonly found in soil and is listed in Table 4.6. As shown in Figure 4.10, SortMeDNA has classified at least 10% more reads than any other tool in the sensitive modes for all tools. Once again, the uniquely found reads by SortMeDNA, SHRiMP2 and Bowtie2 were analyzed with BLASTN to verify their significance (see Table 4.7). At E-value=1e-5, over 72% of reads found only by SortMeDNA are considered highly significant according to BLASTN. As previously mentioned, Ion Torrent reads experience on average 1.5 indels per 100 bases, therefore seeds which can handle indel errors should logically perform better on this type of data. Both SortMeDNA and BWA-SW support indels in their seeds, however the Z-best heuristic in BWA-SW considers a limited number of matching seeds and therefore overlooks important alignments. Increasing the Z-best score from 1 to 100 gains an extra 33% reads for this tool but decreases the speed by a factor of 26, making it the slowest program of the four.

Table 4.5: Substitute species for HMP Mock Community organisms. (The species in **blue** are the original species in the HMP Mock Community samples and species in ⋆**green** are closely related species used to substitute the original ones.)

	Original species	Accession	Substitute species	Accession	Distance matrix value ∈ [0, 1] computed using CVTree [Qi et al., 2004]: 0=identical; 1=unrelated
1	Acinetobacter baumannii	CP000521.1	⋆Acinetobacter calcoaceticus	NC_016603	0.04313
	– plasmid pAB1	NC_009083.1			
	– plasmid pAB2	NC_009084.1			
2	Actinomyces odontolyticus	AAYI02000004.1			
3	Bacillus cereus	AE017194.1	⋆Bacillus pumilus	NC_009848	0.21627
	– plasmid pBc10987	NC_005707.1			
4	Bacteroides vulgatus	CP000139.1	⋆Bacteroides fragilis	NC_006347	0.12134
			– plasmid pBFY46	NC_006297	
5	Candida albicans	SC5314			
6	Clostridium beijerinckii	CP000721.1	⋆Clostridium acetobutylicum	NC_003030	0.20376
			– plasmid pSOL1	NC_001988	
7	Deinococcus radiodurans		⋆Deinococcus geothermalis	NC_008025	0.12772
	– chromosome 1	NC_001263.1	– plasmid pDGEO02	NC_009939	
	– chromosome 2	NC_001264.1	– plasmid pDGEO01	NC_008010	
	– plasmid CP1	NC_000959.1			
	– plasmid MP1	NC_000958.1			
8	Enterococcus faecalis	CP002621.1	⋆Enterococcus casseliflavus	NC_020995	0.20351
9	Escherichia coli	U00096.2			
10	Helicobacter pylori	AE000511.1	⋆Helicobacter cinaedi	NC_017761	0.22390
			– plasmid pHci1	NC_017762	
11	Lactobacillus gasseri	NC_008530.1	⋆Lactobacillus helveticus	NC_010080	0.24046
12	Listeria monocytogenes	NC_003210.1	⋆Listeria marthii	NZ_CM001047	0.22722
13	Methanobrevibacter smithii	CP000678.1	⋆Methanobrevibacter ruminantium	NC_013790	0.27240
14	Neisseria meningitidis	AE002098.2	⋆Neisseria lactamica	NC_014752	0.05882
15	Propionibacterium acnes	NC_006085.1	⋆Propionibacterium propionicum	NC_018142	0.21950
16	Pseudomonas aeruginosa	AE004091.2	⋆Pseudomonas chlororaphis	NZ_CM001490	0.10585
17	Rhodobacter sphaeroides		⋆Rhodobacter capsulatus	NC_014034	0.16064
	– chromosome 1	NC_007493.1	– plasmid pRCB133	NC_014035	
	– chromosome 2	NC_007494.1			
	– plasmid A	NC_009007.1			
	– plasmid B	NC_007488.1			
	– plasmid C	NC_007489.1			
	– plasmid D	NC_007490.1			
	– plasmid E	NC_009008.1			
18	Staphylococcus aureus	NC_010079.1			
	– plasmid pUSA300HOUMR	NC_010063.1			
19	Staphylococcus epidermidis	NC_004461.1			
	– plasmid pSE-122281	NC_005008.1			
	– plasmid pSE-122282	NC_005007.1			
	– plasmid pSE-122283	NC_005006.1			
	– plasmid pSE-122284	NC_005005.1			
	– plasmid pSE-122285	NC_005004.1			
	– plasmid pSE-122286	NC_005003.1			
20	Streptococcus agalactiae	AE009948.1			
21	Streptococcus mutans	AE014133.2			
22	Streptococcus pneumoniae	AE005672.3			

Figure 4.8: Left: Phylogenetic tree for HMP Mock Community samples and their closely related species. The distance matrix was computed with CVTree (the distances shown on the branches and are additive), the tree was created using the neighbor-joining method of the PHYLIP package [Felsenstein, 1989] and drawn with TreeDyn [Chevenet et al., 2006]. The species in **blue** are the original species in the HMP Mock Community samples, and the species in **green** are closely related species used to substitute the original ones. The species in black are those for which no substitute organism was given. **Right**: Box plot illustrating the difference for the distance between original species (**red box**) and between original and subsitute species (**green box**).

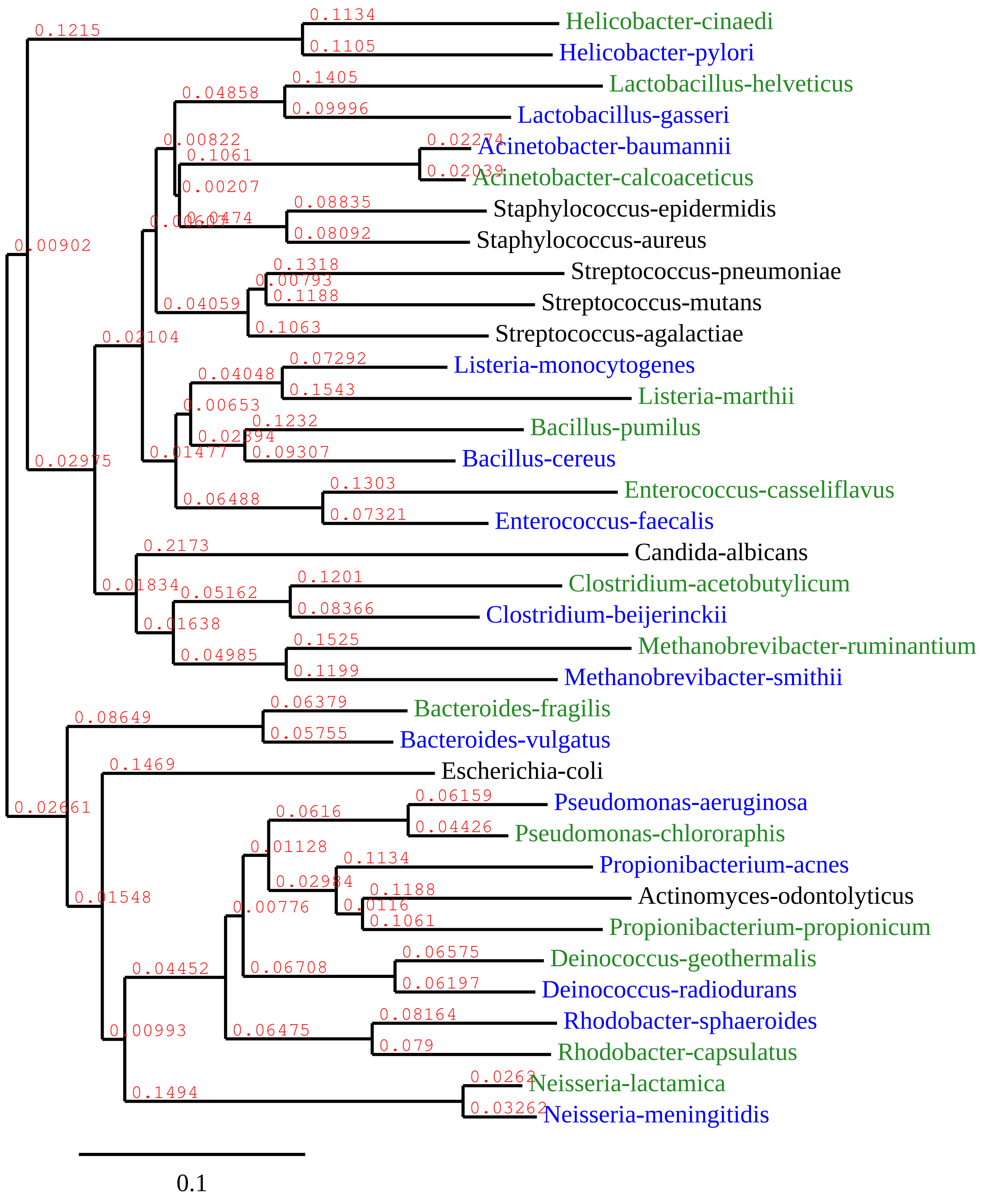


Figure 4.9: Plot illustrating the spread of distance matrix values for original species (**red box**) and for original species and their substitutes (**green box**) as computed by CVTree.

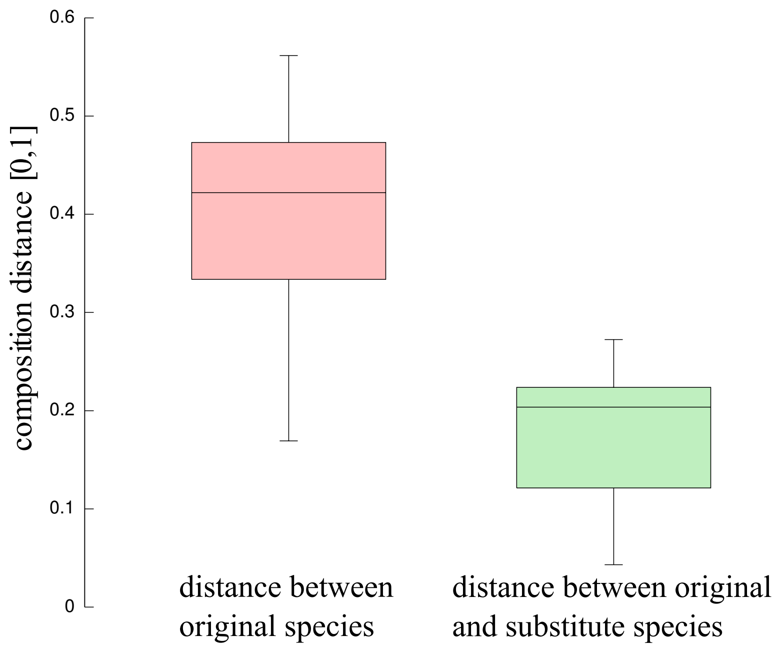
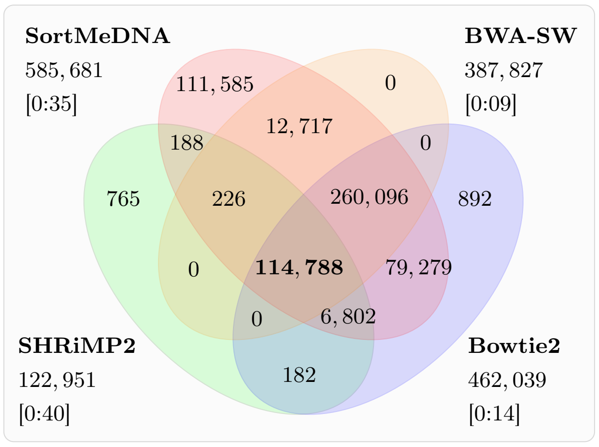


Table 4.6: Bacterial genome reference database for arctic soil metagenome used in Test 6

	Species	Accession
1	*Pseudomonas aeruginosa*	NC_002516.2
2	*Arthrobacter chlorophenolicus*	NC_011886.1
	– plasmid pACHL01	NC_011879.1
	– plasmid pACHL02	NC_011881.1
3	*Clostridium beijerinckii*	NC_009617.1
4	*Enterobacter aerogenes*	NC_020181.1
	– plasmid pEA1509 B	NC_020182.1
	– plasmid pEA1509 A	NC_020180.1
5	*Micrococcus luteus*	NC_012803.1
6	*Corallococcus coralloides*	NC_017030.1
7	*Cytophaga hutchinsonii*	NC_008255.1
8	*Bacillus cereus*	AE016877.1
	– plasmid pBClin15	AE016878.2
9	*Methanococcus voltae*	CP002057.1
10	*Nitrobacter hamburgensis*	CP000319.1
	– plasmid 1	CP000320.1
	– plasmid 2	CP000321.1
	– plasmid 3	CP000322.1
11	*Desulfarculus baarsii*	CP002085.1
12	*Gemmatimonas aurantiaca*	AP009153.1

Figure 4.10: 610,536 Ion Torrent reads (∼210 nt) of arctic soil metagenome against a small 12-species bacterial reference database shown in Table 4.6

in **default** mode:



in **sensitive** mode:

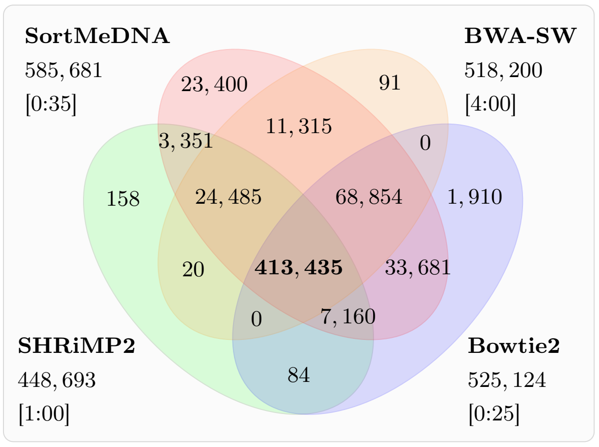


Table 4.7: Analysis of possible false-positive reads found by any one tool in sensitive mode for the arctic soil metagenome (see Venn diagram in Figure 4.10) according to BLASTN. The table shows how many reads found by only one of the tools are also found by BLASTN having identity $\geq 75\%$ and E-value $\in [1, 0.01, 1e\text{-}5, 1e\text{-}10]$.

E-value	SortMeDNA (23,400)		SHRiMP2 (158)		Bowtie2 (1,910)		BWA-SW (91)	
	# reads	% reads	# reads	% reads	# reads	% reads	# reads	% reads
1	22,831	97.5	150	94.9	1,909	99.9	84	92.3
0.01	21,900	93.5	149	94.3	1,905	99.7	78	85.7
1*e*-5	16,911	72.2	98	62.0	753	39.4	78	85.7
1*e*-10	5,479	23.4	26	16.4	0	0.0	54	59.3

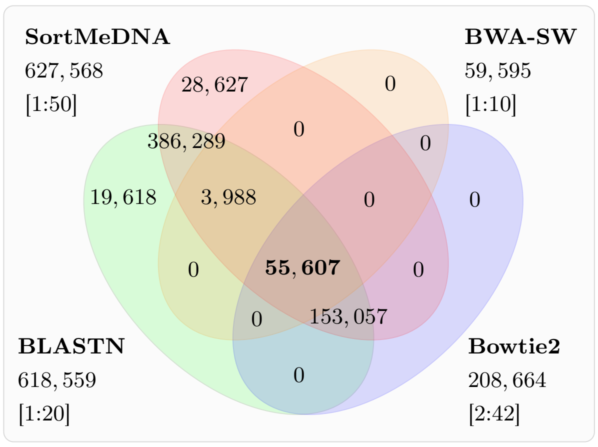
4.3.7 Test 7: PacBio read mapping of Haitian *Vibrio cholerae* strain

PacBio is the only currently available technology to produce read lengths of up to 20 Kbp, directly simplifying sequence comparison for repetitive and low complexity regions. For reference-guided genome reconstruction, long PacBio reads can be used to finish or upgrade an assembly and bridge various gaps caused by these ambiguous regions [English et al., 2012]. However, whereas an Illumina HiSeq 2000 platform can yield up to 3 billion reads per flow cell (HiSeq systems datasheet), a PacBio RS II platform can currently only yield ∼47,000 reads per SMRT cell (PacBio RS II brochure). Morever, the PacBio technology has the highest error rates of roughly 15%, with the majority of errors originating from insertions.

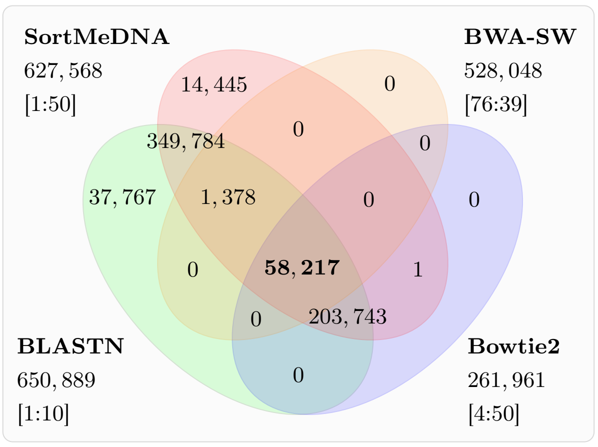
In this experiment we test the capabilities of all tools to align PacBio reads from the bacterial isolate *V. cholerae H1* to the reference strain N16961, taken from a study which originally exposed the origin of the Haitian *V. Cholerae* outbreak in 2011 [Chin et al., 2011]. The original reads file contained 2,911,146 reads, although after filtering to remove reads with <0.75 read quality and <50 nt in length as suggested in the preliminary glossary datasheet by PacBio, the working set narrowed to 807,124 reads. The results are shown in Figure 4.11.

Figure 4.11: 807,124 PacBio reads from *V. cholerae H1* against reference N16961. Default mode for BLASTN sets E-value=1*e*-5 and the sensitive mode sets E-value=0.01

in **default** mode:



in **sensitive** mode:



For this last dataset, we also ran an additional test with Smith-Waterman scores suggested by BWA-SW for PacBio reads (reward = 1, penalty = -5, gap open = -2, gap ext = -1, see note after typing `./bwa bwasw`) with the results shown in Table 4.8.

Table 4.8: 807,124 PacBio reads from *V. cholerae H1* against reference N16961 with dedicated Smith-Waterman score parameters: reward=1, penalty=-5, gap open=-2, gap ext=-1. BLASTN was run with mismatch=-4, since mismatch=-5 was not available with the gap open=-2 and gap extend=-1 settings

	SortMeDNA	BLASTN	BWA-SW	Bowtie2
time [hr:min]	1:46	1:30	10:14	1:00
# reads mapped	631,738	614,204	491,636	198,618

The long read lengths and high error rate showed various effects on different tools. For

SortMeDNA, the execution time remained stable, with 77% of the reads being mapped within 2 hours. BLASTN was the fastest tool, although the reference genome is small (4.1 Mb) and aligning reads to something larger such as the human genome would easily surpass the time for other tools, likewise an increase in cell throughput would render a similar effect. Still, BLASTN mapped between 76-80% of the reads for E-values 1*e*-5 and 0.01, respectively. Using the suggested Smith-Waterman parameters for BWA-SW resulted in 60% of the reads being mapped in 10 hours. In the sensitive mode (with Smith-Waterman parameters set to default of BLASTN), BWA-SW classifies the most reads out of the three settings, although at a much slower speed of 76 hours. The sensitive setting for Bowtie2 mapped only 32% of the reads, less than half of those mapped by SortMeDNA or BLASTN and in nearly the equivalent amount of time. The results for SHRiMP2 were not computed since the runtime exceeded three days with only a small fraction of reads having been processed. As observed in previous tests involving long and length varying reads, in particular for the mapping of 454 reads from the HMP Mock community to original or divergent species, the speed for SHRiMP2 spikes dramatically, where processing less than 1.5 million reads took ∼30 hours compared to ∼2 hours for SortMeDNA (see Figure 4.7).

4.4 Discussion

In the preceding section, we have run a series of experimental tests to assess the accuracy and the efficiency of SortMeDNA compared to its main competitors. Overall, SortMeDNA has shown to be the most robust read mapper for Illumina, 454, Ion Torrent and PacBio reads, with only two intuitive options (`-fast` and `-sensitive`) to regulate mapping for highly similar datasets and more divergent ones. Although SortMeDNA was not optimized for genome resequencing, Test 1 and Test 2 show that its performance remains excellent in this context when using the `-fast` option. In this case, it renders a significant increase in speed (over `-sensitive`) with minimal loss in sensitivity. For more divergent datasets with Tests 3–7, SortMeDNA outperforms the other software, achieving higher sensitivity and maintaining a low computational time. This establishes the relevance of the algorithmic choices behind SortMeDNA.

All the four tools, Bowtie2, Shrimp2, BWA-SW and SortMeDNA take the same overall approach. They firstly locate seeds between the read and the reference database using an indexed copy of the database, and then extend them to longer matches using local Smith-Waterman alignment. The difference mostly lies in the choice for the seed model and for the index data structure. Both Bowtie2 and BWA-SW use an FM-index based on the BWT compression to index the reference database (also the read for BWA-SW). In default mode, Bowtie2 uses exact seeds of length 20 bp and shifts the seed every 8-12 bp on the read (for read lengths 100-250bp). However, options also include allowing one mismatch in the seed, varying seed length and the interval size. BWA-SW uses a dynamic programming algorithm to search for seeds between two FM-indices, implicitly allowing mismatches and gaps. It also uses heuristics to speedup its traversal of the two FM-indices, in default mode making it the fastest alignment tool but also with the least sensitivity. In default mode, SHRiMP2 uses three spaced seeds of weight 12 bp, meaning 12 predetermined positions in the seed must have an exact match (and the remaining can be mismatches), and indexes the reference database using spaced k-mer hashing.

With SortMeDNA, we use novel seeds allowing up to 1 error: The error can either be a mismatch or an indel, and its position in the seed is not predetermined as it is the case with spaced seeds. This unique feature gives the seed flexibility for different error types, such as indels in 454 reads, and unpredictable error distribution, as readily observed with PacBio reads.

However, the tradeoff for maintaining such seeds is the new text index data structure which is able to accomodate searches with insertions and deletions, requiring more space than the BWT. In our implementation, the entries in the leaves of the tries are compressed using 2 bits per nucleotide. Due to the dynamic nature of index construction in SortMeDNA, one cannot precisely calculate the size of the index before it is built. However, from multiple simulations on varying data presented in the performances section, the estimated size has shown to be inferior to 100×(length of reference sequence) bytes. In practice, the index occupies 61 bytes/nt for TAIR10 *A. thaliana* genome and 15 bytes/nt for 80,000 16S rRNA (both about 121Mb). In Test 2, BWA-SW and Bowtie2 were able to index the unmasked human genome in under 6.3Gb, SHRiMP2 split the index into two parts of roughly 20Gb each (32Gb available RAM), and SortMeDNA split the index into 16 parts ranging between 16-21Gb each. As for SHRiMP2, SortMeDNA allows the amount of memory allocated for the index to be regulated through the command line, with the default allowed memory being set to 3Gb. If the index grows beyond this user designated threshold, it is internally split into multiple subparts. During the mapping of reads, each index subpart is sequentially loaded and processed in memory, always maintaining the memory for the index below its designated threshold. Although the optimal mapping time for SortMeDNA is achieved if the index can be constructed in one part, the overhead time with processing multiple subparts is not impractical, as demonstrated in Test 2. The index fragmentation feature nonetheless allows convenient utilization of the tool.

Finally, the statistical E-value is an efficient measure for evaluating alignment scores, and aids to moderate the selectivity and sensitivity results of a software better than the raw Smith-Waterman scores alone, as done in SHRiMP2, BWA-SW and Bowtie2. The E-value parameter will play a significant role in deciding the integrity of an alignment as more and more distant species are aligned, since classifying them is difficult. It should be noted that *without* additional alignment credentials, this classification may change from ‘difficult’ to ‘wrong’, as observed in the sensitivity/selectivity plots in Test 3.

Conclusion

In this thesis, we have introduced a novel seeding technique for approximate seeds and the supporting indexing data structures used to quickly locate short regions of similarity between two sequences. In Chapter 3 we presented SortMeRNA, an efficient filter implementing approximate seeds to quickly remove ribosomal fragments from metatranscriptomic data. Nowadays, SortMeRNA is the fastest algorithm to filter ribosomal RNA with an accuracy comparable to the probabilistic HMM tools. In Chapter 4 we presented SortMeDNA, a read mapping tool that can align genomic and metagenomic data generated by second- and third-generation sequencing. SortMeDNA applies statistical analysis to verify the significance of an alignment since when mapping reads against distantly divergent species the chances of seeing random alignments reaching a threshold score increase.

Advancements in DNA sequencing techniques will one day provide technologies that skip the whole-genome shotgunning step and sequence complete genomes in one go. This trend is already visible with the 4,000-20,000 bp reads produced by PacBio, and entire DNA strand sequencing for the Lambda phage genome of 48,000 bp (both strands) by GridION [GridION, 2012]. However, although short read mapping tools may no longer be necessary, the algorithms behind them can prove to be invaluable for other sequence analysis applications. Genetic sequence databases such as GenBank [Benson et al., 2005], RefSeq [Pruitt et al., 2009] and Ensembl [Flicek et al., 2013] are growing at an expontential rate, with GenBank storing over 150 billion nucleotide bases in August, 2013 (see GenBank release notes for release 197.0). What is more, the databases are composed of a variety of sequences including whole genomes, metagenomic sequences, ribosomal, chloroplast, mitochondrial and uncultured sequences. Sorting through and analyzing all of this data can only be done by extremely efficient software and large computational resources. Finding DNA sequence variations [Cheng et al., 2013], making genetic diagnoses of diseases, studying microbial pathogens responsible for crop losses [Newton et al., 2010] and plant biology, as well as understanding the vast network and interdependancy of environmental communities [Fierera et al., 2012] are just a few possibilities of where new bioinformatic analysis tools can evolve. In the meantime, as high-throughput short read technologies still dominate the market, new software tools capable of quickly and accurately analyzing their data are essential.

Both of our tools offer a good balance between speed and sensitivity, which is important for the analysis of high-throughput data for new applications in metatranscriptomics and metagenomics. The compromise to this feature is the much larger index required for implementing an approximate seed search, especially for large and repetitive eukaryotic genomes. Since SortMeRNA works with highly conserved ribosomal sequences, it requires a small representative database to achieve a very high sensitivity. By using either the 8,000 16S rRNA representative sequences provided with the SortMeRNA distribution or the 400,000 16S rRNA sequences available in the original SILVA databases, the differences in the number of reads classified by SortMeRNA will be very small. SortMeDNA on the other hand is capable of managing the amount of memory loaded in RAM by internally splitting the index into multiple subparts. Investigations into reducing the size of the index are important to consider for future improvements of both tools. Perhaps a compressed version of the burst trie can be devised, or a completely different data structure similar to an enhanced suffix array or the BWT can be adopted. Compressed full-text indices such as the BWT or the compressed suffix array may deliver the most promising algorithms for future generation software as they are able to simultaneously compress and index a text [Navarro and Mäkinen, 2007]. It was also shown in [Russo et al., 2009] that it is possible to perform approximate string matching (up to $d = 6$) using compressed indices such as the Lempel-Ziv and suffix array self indices.

A possible extension to the seed model could be adding weights to the transitions of the universal Levenshtein automaton. Currently, seeds having a mismatch error or an indel error are scored equally, however as in the Smith-Waterman scoring matrix, options giving different weights to mismatches and indels can influence the results (for example fewer false positives by filtering out low probability matches). Furthermore, having a weighted automaton could also be exploited for seeds allowing 2 or even 3 errors. Although these latter options are not yet available in our tools, they could prove to be useful than the 1-error model in studies involving many low-complexity sequences or to discover more distant species.

Bibliography

[Abouelhoda et al., 2004] Abouelhoda, M., Kurtz, S., and Ohlebusch, E. (2004). Replacing suffix trees with enhanced suffix arrays. *J Discrete Algorithms*, 2:53–86.

[Agrawal and Huang, 2008] Agrawal, A. and Huang, X. (2008). DNAlignTT: pairwise DNA alignment with sequence specific transition-transversion ratio. In *IEEE Intl Conf EIT*, pages 457–459.

[Ahmed et al., 2004] Ahmed, A., Bararia, D., Vinayak, S., Yameen, M., Biswas, S., Dev, V., Kumar, A., Ansari, M., and Sharma, Y. (2004). Plasmodium falciparum isolates in India exhibit a progressive increase in mutations associated with sulfadoxine-pyrimethamine resistance. *Antimicrob Agents Chemother*, 48:879–889.

[Altschul and Gish, 1996] Altschul, S. and Gish, W. (1996). Local alignment statistics. *Methods Enzymol*, 266:460–480.

[Altschul et al., 1990] Altschul, S. F., Gish, W., Miller, W., Myers, E. W., and Lipman, D. J. (1990). Basic Local Alignment Search Tool. *J Mol Biol*, 215:403–410.

[Benson et al., 2005] Benson, D., Karsch-Mizrachi, I., Lipman, D., Ostell, J., and Wheeler, D. (2005). GenBank. *Nucl Acids Res*, 33:D34–D38.

[Bentley et al., 2008] Bentley, D. R., Balasubramanian, S., Swerdlow, H. P., Smith, G. P., Milton, J., Brown, C. G., Hall, K. P., Evers, D. J., Barnes, C. L., Bignell, H. R., Boutell, J. M., Bryant, J., Carter, R. J., Keira Cheetham, R., Cox, A. J., Ellis, D. J., Flatbush, M. R., Gormley, N. A., Humphray, S. J., Irving, L. J., Karbelashvili, M. S., Kirk, S. M., Li, H., Liu, X., Maisinger, K. S., Murray, L. J., Obradovic, B., Ost, T., Parkinson, M. L., Pratt, M. R., Rasolonjatovo, I. M. J., Reed, M. T., Rigatti, R., Rodighiero, C., Ross, M. T., Sabot, A., Sankar, S. V., Scally, A., Schroth, G. P., Smith, M. E., Smith, V. P., Spiridou, A., Torrance, P. E., Tzonev, S. S., Vermaas, E. H., Walter, K., Wu, X., Zhang, L., Alam, M. D., Anastasi, C., Aniebo, I. C., Bailey, D. M. D., Bancarz, I. R., Banerjee, S., Barbour, S. G., Baybayan, P. A., Benoit, V. A., Benson, K. F., Bevis, C., Black, P. J., Boodhun, A., Brennan, J. S., Bridgham, J. A., Brown, R. C., Brown, A. A., Buermann, D. H., Bundu, A. A., Burrows, J. C., Carter, N. P., Castillo, N., Chiara E. Catenazzi, M., Chang, S., Neil Cooley, R., Crake, N. R., Dada, O. O., Diakoumakos, K. D., Dominguez-Fernandez, B., Earnshaw, D. J., Egbujor, U. C., Elmore, D. W., Etchin, S. S., Ewan, M. R., Fedurco, M., Fraser, L. J., Fuentes Fajardo, K. V., Scott Furey, W., George, D., Gietzen, K. J., Goddard, C. P., Golda, G. S., Granieri, P. A., Green, D. E., Gustafson, D. L., Hansen, N. F., Harnish, K., Haudenschild, C. D., Heyer, N. I., Hims, M. M., Ho, J. T., Horgan, A. M., Hoschler, K., Hurwitz, S., Ivanov, D. V., Johnson, M. Q., James, T., Huw Jones, T. A., Kang, G.-D., Kerelska, T. H., Kersey, A. D., Khrebtukova, I., Kindwall, A. P., Kingsbury, Z., Kokko-Gonzales, P. I., Kumar, A., Laurent, M. A., Lawley, C. T., Lee, S. E., Lee, X., Liao, A. K., Loch, J. A., Lok, M., Luo, S., Mammen, R. M., Martin, J. W., McCauley, P. G., McNitt, P., Mehta, P., Moon, K. W., Mullens, J. W., Newington, T., Ning, Z., Ling Ng, B., Novo, S. M., O'Neill, M. J., Osborne, M. A., Osnowski, A., Ostadan, O., Paraschos, L. L., Pickering, L., Pike, A. C., Pike, A. C., Chris Pinkard, D., Pliskin, D. P., Podhasky, J., Quijano, V. J., Raczy, C., Rae, V. H., Rawlings, S. R., Chiva Rodriguez, A., Roe, P. M., Rogers, J., Rogert Bacigalupo, M. C., Romanov, N., Romieu, A., Roth, R. K., Rourke, N. J., Ruediger, S. T., Rusman, E., Sanches-Kuiper, R. M., Schenker, M. R., Seoane, J. M., Shaw, R. J., Shiver, M. K., Short, S. W., Sizto, N. L., Sluis, J. P., Smith, M. A., Ernest Sohna Sohna, J., Spence, E. J., Stevens, K., Sutton, N., Szajkowski, L., Tregidgo, C. L., Turcatti, G., vandeVondele, S., Verhovsky, Y., Virk, S. M., Wakelin, S., Walcott, G. C., Wang, J., Worsley, G. J., Yan, J., Yau, L., Zuerlein, M., Rogers, J., Mullikin, J. C., Hurles, M. E., McCooke, N. J., West, J. S., Oaks, F. L., Lundberg, P. L., Klenerman, D., Durbin, R., and Smith, A. J. (2008). Accurate whole human genome sequencing using reversible terminator chemistry. *Nature*, 456:53–59.

[Boissinot et al., 2007] Boissinot, K., Huletsky, A., Peytavi, R., Turcotte, S., Veillette, V., Boissinot, M., Picard, F., Martel, E., and Bergeron, M. (2007). Rapid exonuclease digestion of PCR-amplified targets for improved microarray hybridization. *Clin Chem*, 53:2020–2023.

[Boyce et al., 2010] Boyce, D. G., Lewis, M. R., and Worm, B. (2010). Global phytoplankton decline over the past century. *Nature*, 466:591–596.

[Buckwalter et al., 2013] Buckwalter, P., Embaye, T., Gormly, S., and Trent, J. D. (2013). Dewatering microalgae by forward osmosis. *Desalination*, 312:19–22.

[Burrows and Wheeler, 1994] Burrows, M. and Wheeler, D. (1994). A block-sorting lossless data compression algorithm. Technical report.

[Cannone et al., 2002] Cannone, J., Subramanian, S., Schnare, M., Collett, J., D'Souza, L., and Du, Y. (2002). The comparative RNA web (CRW) site: An online database of comparative sequence and structure information for ribosomal, intron, and other RNAs. *BMC Bioinformatics*, 3:15.

[Chacón et al., 2013] Chacón, A., Moure, J.-C., Espinosa, A., and Hernández, P. (2013). *n*-step FM-index for faster pattern matching. In *International Conference on Computational Science, ICCS*, pages 70–79.

[Chan et al., 2013] Chan, Y., Van Nostrand, J., Zhou, J., Pointing, S., and Farrell, R. (2013). Functional ecology of an Antarctic Dry Valley. *Proc Natl Acad Sci*, 110:8990–8995.

[Cheng et al., 2013] Cheng, L., Connor, T., Sirén, J., Aanensen, D., and Corander, J. (2013). Hierarchical and spatially explicit clustering of DNA sequences with BAPS software. *Mol Biol Evol*, 30:1224–1228.

[Chevenet et al., 2006] Chevenet, F., Brun, C., Banuls, A., Jacq, B., and Christen, R. (2006). Treedyn: towards dynamic graphics and annotations for analyses of trees. *BMC Bioinformatics*, 7.

[Chiaromonte et al., 2002] Chiaromonte, F., Yap, V., and Miller, W. (2002). Scoring pairwise genomic sequence alignments. *Pac Symp Biocomput*, pages 115–26.

[Chin et al., 2011] Chin, C. S., Sorenson, J., Harris, J. B., Robins, W. P., Charles, R. C., Jean-Charles, R. R., Bullard, J., Webster, D. R., Kasarskis, A., Peluso, P., Paxinos, E. E., Yamaichi, Y., Calderwood, S. B., Mekalanos, J. J., Schadt, E. E., and Waldor, M. K. (2011). The origin of the haitian cholera outbreak strain. *New England Journal of Medicine*, 364:33–42.

[Clement et al., 2010] Clement, N., Snell, Q., Clement, M., Hollenhorst, P., Purwar, J., Graves, B., Cairns, B., and Johnson, E. (2010). The GNUMAP algorithm: unbiased probabilistic mapping of oligonucleotides from next-generation sequencing. *Bioinformatics*, 26:38–45.

[Consortium, 2001] Consortium, I. H. G. S. (2001). Initial sequencing and analysis of the human genome. *Nature*, 409:860–921.

[David et al., 2011] David, M., Dzamba, M., Lister, D., Ilie, L., and Brudno, M. (2011). SHRiMP2: Sensitive yet practical short read mapping. *Bioinformatics*, 27:1011–1012.

[Dayhoff and Schwartz, 1978] Dayhoff, M. O. and Schwartz, R. M. (1978). Chapter 22: A model of evolutionary change in proteins. In *Atlas of Protein Sequence and Structure*.

[Dumas and Ninio, 1982] Dumas, J. and Ninio, J. (1982). Efficient algorithms for folding and comparing nucleic acid sequences. *Nucl Acids Res*, 10:197–206.

[Durbin et al., 1998] Durbin, R., Eddy, S., Krogh, A., and Mitchison, G. (1998). *Biological sequence analysis: probabilistic models of proteins and nucleic acids*, chapter Pairwise alignment. Cambridge University Press.

[Eddy, 2011] Eddy, S. R. (2011). Accelerated profile HMM searches. *PLoS Comput Biol*, 7(10):e1002195.

[Edgar, 2010] Edgar, R. (2010). Search and clustering orders of magnitude faster than blast. *Bioinformatics*, 26(19):2460–2461.

[Ehrlich, 2011] Ehrlich, S. D. (2011). *MetaHIT: The European Union Project on Metagenomics of the Human Intestinal Tract*, pages 307–316. Springer, London.

[English et al., 2012] English, A. C., Richards, S., Han, Y., Wang, M., Vee, V., Qu, J., Qin, X., Muzny, D. M., Reid, J. G., Worley, K. C., and Gibbs, R. A. (2012). Mind the gap: upgrading genomes with pacific biosciences rs long-read sequencing technology. *PLoS ONE*, 7:doi:10.1371/journal.pone.0047768.

[Farach, 1997] Farach, M. (1997). Optimal suffix tree construction with large alphabets. In *38th IEEE Symposium on Foundations of Computer Science*, pages 137–143.

[Farrar, 2007] Farrar, M. (2007). Striped Smith-Waterman speeds database searches six times over other SIMD implementations. *Bioinformatics*, 23:156–161.

[Felsenstein, 1989] Felsenstein, J. (1989). Phylip - phylogeny inference package. *Cladistics*, 5:164–166.

[Ferragina and Manzini, 2000] Ferragina, P. and Manzini, G. (2000). Opportunistic data structures with applications. *FOCS '00 Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, pages 390–398.

[Fierera et al., 2012] Fierera, N., Leff, J., Adams, B., Nielsen, U., Bates, S., Lauber, C., Owense, S., Gilbert, J., Wall, D., and Caporaso, J. (2012). Cross-biome metagenomic analyses of soil microbial communities and their functional attributes. *Proc Natl Acad Sci*, 109:doi:10.1073/pnas.1215210110.

[Fleischmann et al., 1995] Fleischmann, R., Adams, M., White, O., Clayton, R., Kirkness, E., Kerlavage, A., Bult, C., Tomb, J., Dougherty, B., and Merrick, J.M., e. a. (1995). Whole-genome random sequencing and assembly of Haemophilus influenzae rd. *Science*, 269:496–512.

[Flicek et al., 2013] Flicek, P., Ahmed, I., Amode, M., Barrell, D., Beal, K., Brent, S., Carvalho-Silva, D., Clapham, P., Coates, G., Fairley, S., Fitzgerald, S., Gil, L., Garcia-Giron, C., Gordon, L., Hourlier, T., Hunt, S., Juettemann, T., Kähäri, A., Keenan, S., Komorowska, M., Kulesha, E., Longden, I., Maurel, T., McLaren, W., Muffato, M., Nag, R., Overduin, B., Pignatelli, M., Pritchard, B., Pritchard, E., Riat, H., Ritchie, G., Ruffier, M., Schuster, M., Sheppard, D., Sobral, D., Taylor, K., Thormann, A., Trevanion, S., White, S., Wilder, S., Aken, B., Birney, E., Cunningham, F., Dunham, I., Harrow, J., Herrero, J., Hubbard, T., Johnson, N., Kinsella, R., Parker, A., Spudich, G., Yates, A., Zadissa, A., and Searle, S. (2013). GenBank. *Nucl Acids Res*, 41:D48–D55.

[Fonseca et al., 2010] Fonseca, V., Carvalho, G., Sung, W., Johnson, H., Power, D., Neill, S., Packer, M., Blaxter, M., Lambshead, P., Thomas, W., and Creer, S. (2010). Second-generation environmental sequencing unmasks marine metazoan biodiversity. *Nat Commun*, 1:doi:10.1038/ncomms1095.

[Frith et al., 2010] Frith, M. C., Hamada, M., and Horton, P. (2010). Parameters for accurate genome alignment. *BMC Bioinformatics*, 11.

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

[Goffeau et al., 1996] Goffeau, A., Barrell, B., Bussey, H., Davis, R., Dujon, B., Feldmann, H., Galibert, F., Hoheisel, J., Jacq, C., Johnston, M., Louis, E., Mewes, H., Murakami, Y., Philippsen, P., Tettelin, H., and Oliver, S. (1996). Life with 6000 genes. *Science*, 274:563–567.

[Gonnet et al., 1992] Gonnet, G., Baeza-Yates, R., and Snider, T. (1992). New indices for text: PAT trees and PAT arrays. In *Information Retrieval: Data Structures and Algorithms*, chapter 5, pages 66–82.

[Gordon et al., 1986] Gordon, L., Schilling, M. F., and Waterman, M. S. (1986). An extreme value theory for long head runs. *Probab Th Rel Fields*, 72:279–287.

[Gotoh, 1982] Gotoh, O. (1982). An improved algorithm for matching biological sequences. *J Mol Biol*, 162:705–708.

[GridION, 2012] GridION, O. N. (2012). Oxford nanopore introduces DNA 'strand sequencing' on the high-throughput GridION platform and presents MinION, a sequencer the size of a USB memory stick. https://www.nanoporetech.com/news/press-releases/view/39. [Online; accessed 10-Oct-2013].

[Gusfield, 1997] Gusfield, D. (1997). *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, New York.

[Hatem et al., 2013] Hatem, A., Bozda, D., Toland, A., and Catalyurek, U. (2013). Benchmarking short sequence mapping tools. *BMC Bioinformatics*, 14:doi:10.1186/1471–2105–14–184.

[Heinz et al., 2002] Heinz, S., Zobel, J., and Williams, H. E. (2002). Burst tries: A fast, efficient data structure for string keys. *ACM Transactions on Information Systems*, 20:192–223.

[Henikoff and Henikoff, 1992] Henikoff, S. and Henikoff, J. G. (1992). Amino acid substitution matrices from protein blocks. In *Proc Natl Acad Sci*, volume 89.

[Heppner, 1923] Heppner, M. J. (1923). The factor for bitterness in the sweet almond. *Genetics*, 8:390–391.

[Hingamp et al., 2013] Hingamp, P., Grimsley, N., Acinas, S. G., Clerissi, C., Subirana, L., Poulain, J., Ferrera, I., Sarmento, H., Villar, E., Lima-Mendez, G., Faust, K., Sunagawa, S., Claverie, J. M., Moreau, H., Desdevises, Y., Bork, P., Raes, J., de Vargas, C., Karsenti, E., Kandels-Lewis, S., Jaillon, O., Not, F., Pesant, S., Wincker, P., and Ogata, H. (2013). Exploring nucleo-cytoplasmic large DNA viruses in Tara Oceans microbial metagenomes. *ISME J*, page doi: 10.1038/ismej.2013.59.

[Hoffmann et al., 2009] Hoffmann, S., Otto, C., Kurtz, S., Sharma, C., Khaitovich, P., Vogel, J., Stadler, P., and Hackermüller, J. (2009). Fast mapping of short sequences with mismatches, insertions and deletions using index structures. *PLoS Comput Biol*, 5:doi:10.1371/journal.pcbi.1000502.

[Holtgrewe, 2010] Holtgrewe, M. (2010). Mason - a read simulator for second generation sequencing data. *Technical Report TR-B-10-06*.

[Holtgrewe et al., 2011] Holtgrewe, M., Emde, A. K., Weese, D., and Reinert, K. (2011). A novel and well-defined benchmarking method for second generation read mapping. *BMC Bioinformatics*, 12:doi:10.1186/1471–2105–12–21.

[Homer et al., 2009] Homer, N., Merriman, B., and Nelson, S. (2009). BFAST: An alignment tool for large scale genome resequencing. *PLoS ONE*, 4:doi:10.1371/journal.pone.0007767.

[Hopcroft et al., 2004] Hopcroft, J., Motwani, R., and Ullman, J. (2004). *Chapter 2.3.5 Equivalence of deterministic and nondeterministic finite automata*. Pearson Education, Boston, MA, 3 edition.

[Huang et al., 2009] Huang, Y., Gilna, P., and Li, W. (2009). Identification of ribosomal RNA genes in metagenomic fragments. *Bioinformatics*, 25:1338–1340.

[Iengar, 2012] Iengar, P. (2012). An analysis of substitution, deletion and insertion mutations in cancer genes. *Nucl Acids Res*, 40:6401–6413.

[Jabbari and Bernardi, 2004] Jabbari, K. and Bernardi, G. (2004). Cytosine methylation and CpG, TpG (CpA) and TpA frequencies. *Gene*, 26:143–149.

[Janda and Abbott, 2007] Janda, J. and Abbott, S. (2007). 16s rRNA gene sequencing for bacterial identification in the diagnostic laboratory: Pluses, perils, and pitfalls. *J Clin Microbiol*, 45:2761–2764.

[Kallmeyer et al., 2012] Kallmeyer, J., Pockalny, R., Adhikari, R. R., Smith, D. C., and D'Hondt, S. (2012). Global distribution of microbial abundance and biomass in subseafloor sediment. *Proc Natl Acad Sci*, 109:16213–16216.

[Karlin and Altschul, 1990] Karlin, S. and Altschul, S. F. (1990). Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proc Natl Acad Sci*, 87:2264–2268.

[Khosravi et al., 2013] Khosravi, Y., Rehvathy, V., Wee, Y., Wang, S., Baybayan, P., Singh, S., Ashby, M., Ong, J., Amoyo, A., Seow, S., Choo, S., Perkins, T., Chua, E., Tay, A., Marshall, B., Loke, M., Goh, K., Pettersson, S., and Vadivelu, J. (2013). Comparing the genomes of *helicobacter pylori* clinical strain UM032 and Mice-adapted derivatives. *Gut Pathogens*, 5:doi:10.1186/1757–4749–5–25.

[Kim and Kececioglu, 2007] Kim, E. and Kececioglu, J. (2007). Inverse sequence alignment from partial examples. In *Algorithms in Bioinformatics, 7th International Workshop (WABI)*, pages 359–370.

[Kopylova et al., 2013] Kopylova, E., Noé, L., Da Silva, C., Berthelot, J.-F., Alberti, A., Aury, J.-M., and Touzet, H. (2013). Deciperhing metatranscriptomic data. In Picardi, E., editor, *Methods in Molecular Biology*, page (to appear).

[Kopylova et al., 2012] Kopylova, E., Noé, L., and Touzet, H. (2012). SortMeRNA: fast and accurate filtering of ribosomal RNAs in metatranscriptomic data. *Bioinformatics*, 28:3211–3217.

[Korf et al., 2003] Korf, I., Yandell, M., and Bedell, J. (2003). *Chapter 4 Sequence Similarity*, pages 55–71. O'Reilly, Sebastopol, CA.

[Krohn-Molt et al., 2013] Krohn-Molt, I., Wemheuer, B., Alawi, M., Poehlein, A., Güllert, S., Schmeisser, C., Pommerening-Röser, A., Grundhoff, A., Daniel, R., Hanelt, D., and Streit, W. R. (2013). Metagenome survey of a multispecies and algae-associated biofilm reveals key elements of bacterial-algae interactions in photobioreactors. *Applied and Environmental Microbiology*.

[Kumar and Filipski, 2007] Kumar, S. and Filipski, A. (2007). Multiple sequence alignment: In pursuit of homologous DNA positions. *Genome Res*, 17:127–135.

[Kurtz, 1999] Kurtz, S. (1999). Reducing the space requirement of suffix trees. *Softw Pract Exper*, 29:1149–1171.

[Kurtz et al., 2004] Kurtz, S., Phillippy, A., Delcher, A., Smoot, M., Shumway, M., Antonescu, C., and Salzberg, S. (2004). Versatile and open software for comparing large genomes. *Genome Biol*, 5:doi:10.1186/gb–2004–5–2–r12.

[Kypr et al., 1989] Kypr, J., Mrázek, J., and Reich, J. (1989). Nucleotide composition bias and CpG dinucleotide content in the genomes of HIV and HTLV 1/2. *Biochimica et Biophysica Acta*, 1009:280–282.

[Lam et al., 2008] Lam, T., Sung, W., Tam, S., Wong, C., and Yiu, S. (2008). Compressed indexing and local alignment of DNA. *Bioinformatics*, 24:791–797.

[Langmead and Salzberg, 2012] Langmead, B. and Salzberg, S. (2012). Fast gapped-read alignment with bowtie 2. *Nat Methods*, 9:357–359.

[Langmead et al., 2009] Langmead, B., Trapnell, C., Pop, M., and Salzberg, S. (2009). Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol*, 10:doi:10.1186/gb–2009–10–3–r25.

[Lee et al., 2011] Lee, J., Yi, H., and Chun, J. (2011). rRNAselector: a computer program for selecting ribosomal RNA encoding sequences from metagenomic and metatranscriptomic shotgun libraries. *J Microbiol*, 49:689–91.

[Leimena et al., 2013] Leimena, M., Ramiro-Garcia, J., Davids, M., van den Bogert, B., Smidt, H., Smid, E., Boekhorst, J., Zoetendal, E., Schaap, P., and Kleerebezem, M. (2013). A comprehensive metatranscriptome analysis pipeline and its validation using human small intestine microbiota datasets. *BMC Genomics*, 14(1):530.

[Lemay et al., 2013] Lemay, M., Henry, P., Lamb, C., Robson, K., and M.A., R. (2013). Novel genomic resources for a climate change sensitive mammal: characterization of the American pika transcriptome. *BMC Genomics*, 14:doi: 10.1186/1471–2164–14–311.

[Letsch et al., 2010] Letsch, H. O., Kück, P., Stocsits, R., and Misof, B. (2010). The impact of rRNA secondary structure consideration in alignment and tree reconstruction: Simulated data and a case study on the phylogeny of hexapods. *Mol Biol Evol*, 27:2507–2521.

[Levene et al., 2003] Levene, M., Korlach, J., Turner, S., Foquet, M., Craighead, H., and Webb, W. (2003). Zero-mode waveguides for single-molecule analysis at high concentrations. *Science*, 5607:682–686.

[Li, 2012] Li, H. (2012). Whole genome simulation. http://sourceforge.net/apps/mediawiki/dnaa/index.php?title=Whole_Genome_Simulation.

[Li and Durbin, 2009] Li, H. and Durbin, R. (2009). Fast and accurate short read alignment with Burrows-Wheeler Transform. *Bioinformatics*, 25:1754–60.

[Li and Homer, 2010] Li, H. and Homer, N. (2010). A survey of sequence alignment algorithms for next-generation sequencing. *Briefings in Bioinformatics*, 11:473–483.

[Li et al., 2008] Li, H., Ruan, J., and Durbin, R. (2008). Mapping short DNA sequencing reads and calling variants using mapping quality scores. *Genome Res*, 18:1851–1858.

[Li et al., 2009] Li, R., Yu, C., and Li, Y. (2009). SOAP2: an improved ultrafast tool for short read alignment. *Bioinformatics*, 25:1966–1967.

[Lipman and Pearson, 1985] Lipman, D. and Pearson, W. (1985). Rapid and sensitive protein similarity searches. *Science*, 227:1435–1441.

[Lisi, 2007] Lisi, M. (2007). Some remarks on the cantor pairing function. *Le Matematiche*, LXII:55–65.

[Loman et al., 2012] Loman, N., Misra, R., Dallman, T., Constantinidou, C., Gharbia, S., Wain, J., and Pallen, M. (2012). Performance comparison of benchtop high-throughput sequencing platforms. *Nat Biotechnol*, 30:434–439.

[Ludwig et al., 2004] Ludwig, W., Strunk, O., Westram, R., Richter, L., Meier, H., Buchner, A., Lai, T., Foè, W., Brettske, I., Gerber, S., Ginhart, A., Gross, O., Grumann, S., Hermann, S., Jost, R., Nig, A., Liss, T., Ûmann, R., May, M., Nonhoff, B., Reichel, B., Strehlow, R., Stamatakis, R., Stuckmann, N., Vilbig, E., Lenke, M., Ludwig, T., and Bode, A. (2004). ARB: A software environment for sequence data. *Nucl Acids Res*, 32:1363–1371.

[Lupski and Stankiewics, 2005] Lupski, J. and Stankiewics, P. (2005). Genomic disorders: Molecular mechanisms for rearrangements and conveyed phenotypes. *PLoS Genet*, 1:e49.

[Ma et al., 2002] Ma, B., Tromp, J., and Li, M. (2002). PatternHunter: faster and more sensitive homology search. *Bioinformatics*, 18:440–445.

[Ma et al., 2013] Ma, C.-Y., Lin, S.-H., Lee, C.-C., Tang, C. Y., Berger, B., and Liao, C.-S. (2013). Reconstruction of phyletic trees by global alignment of multiple metabolic networks. *BMC Bioinformatics*, 14:S12.

[Manber and Myers, 1993] Manber, U. and Myers, E. (1993). Suffix arrays: a new method for on-line string searches. *SIAM J Comput*, 22:935–948.

[Marco-Sola et al., 2012] Marco-Sola, S., Sammeth, M., Guigó, R., and Ribeca, P. (2012). The GEM mapper: fast, accurate and versatile alignment by filtration. *Nat Methods*, 9:1185–1188.

[Mashayekhi and Ronaghi, 2007] Mashayekhi, F. and Ronaghi, M. (2007). Analysis of read length limiting factors in pyrosequencing chemistry. *Anal Biochem*, 363:275–287.

[McCreight, 1976] McCreight, E. (1976). A space-economical suffix tree construction algorithm. *J ACM*, 23:262–272.

[Mears et al., 2002] Mears, J., Cannone, J., Stagg, S., Gutell, R., Agrawal, R., and Harvey, S. (2002). Modeling a minimal ribosome based on comparative sequence analysis. *J Mol Biol*, 321:215–34.

[Meek et al., 2003] Meek, C., Patel, J. M., and Kasetty, S. (2003). OASIS: An online and accurate technique for local-alignment searches on biological sequences. In *In VLDB*, pages 910–921.

[Mihov and Schulz, 2004] Mihov, S. and Schulz, K. (2004). Fast approximate search in large dictionaries. *J Comput Ling*, 30:451–477.

[Mitankin, 2005] Mitankin, P. (2005). Universal Levenshtein automata. building and properties. Master's thesis, Sofia University, Bulgaria.

[Morozova et al., 2007] Morozova, D., Mohlmann, D., and Wagner, D. (2007). Survival of methanogenic archaea from siberian permafrost under simulated martian thermal conditions. *Orig Life Evol Biosph*, 37:189–200.

[Nakamura et al., 2011] Nakamura, K., Oshima, T., Morimoto, T., Ikeda, S., Yoshikawa, H., Shiwa, Y., Ishikawa, S., Linak, M., Hirai, A., Takahashi, H., Altaf-Ul-Amin, M., Ogasawara, N., and Kanaya, S. (2011). Sequence-specific error profile of Illumina sequencers. *Nucl Acids Res*, 39:doi: 10.1093/nar/gkr344.

[Nakamura et al., 2008] Nakamura, S., Maeda, N., Miron, I., Yoh, M., Izutsu, K., Kataoka, C., Honda, T., Yasunaga, T., Nakaya, T., Kawai, J., Hayashizaki, Y., Horii, T., and Iida, T. (2008). Metagenomic diagnosis of bacterial infections. *Emerg Infect Dis*, 14:1784–1786.

[Navarro and Baeza-Yates, 2000] Navarro, G. and Baeza-Yates, R. (2000). A hybrid indexing method for approximate string matching. *J of Discrete Algorithms*, 1:205–239.

[Navarro and Mäkinen, 2007] Navarro, G. and Mäkinen, V. (2007). Compressed full-text indexes. *ACM Comput Surv*, 39:doi:10.1145/1216370.1216372.

[Nawrocki et al., 2009] Nawrocki, E., Kolbe, D., and Eddy, S. (2009). Infernal 1.0: inference of RNA alignments. *Bioinformatics*, 25:1335–7.

[Needleman and Wunsch, 1970] Needleman, S. and Wunsch, C. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J Mol Biol*, 48:443–453.

[Newton et al., 2010] Newton, A., Fitt, B., Atkins, S., Walters, D., and Daniell, T. (2010). Pathogenesis, parasitism and mutualism in the trophic space of microbe-plant interactions. *Trends Microbiol*, 18:365–373.

[Nicholsona et al., 2013] Nicholsona, W., Leonarda, M., Fajardo-Cavazosa, P., Panayotovab, N., Farmerieb, W., Tripletta, E., and Schuergerc, A. (2013). Complete genome sequence of *serratia liquefaciens* strain ATCC 27592. *Genome Announc*, 1:doi: 10.1128/genomeA.00548–13.

[Nyrén, 2007] Nyrén, P. (2007). The history of pyrosequencing. *Methods Mol Biol*, 373:1–14.

[Nystedt et al., 2013] Nystedt, B., Street, N. R., Wetterbom, A., Zuccolo, Lin, Y. C., Scofield, D. G., Vezzi, F., Delhomme, N., Giacomello, S., Alexeyenko, A., Vicedomini, R., Sahlin, K., Sherwood, E., Elfstrand, M., Gramzow, L., Holmberg, K., Hallman, J., Keech, O., Klasson, L., Koriabine, M., Kucukoglu, M., Kaller, M., Luthman, J., and Lysholm (2013). The Norway spruce genome sequence and conifer genome evolution. *Nature*, 497:579–584.

[Olsen et al., 1999] Olsen, R., Bundschuh, R., and Hwa, T. (1999). Rapid assessment of extremal statistics for gapped local alignment. *Proc Int Conf Intell Syst Mol Biol*, pages 211–222.

[Ouyang and Buell, 2004] Ouyang, S. and Buell, C. R. (2004). The TIGR plant repeat databases: a collective resource for the identification of repetitive sequences in plants. *Nucl Acids Res*, 32:D360–363.

[Park et al., 2009] Park, Y., Sheetlin, S., and Spouge, J. L. (2009). Estimating the Gumbel scale parameter for local alignment of random sequences by importance sampling with stopping times. *Ann Stat*, 37:3697.

[Pearson, 1991] Pearson, W. (1991). Searching protein sequence libraries: Comparison of the sensitivity and selectivity of the Smith-Waterman and FASTA algorithms. *Genomics*, 11:635–650.

[Pearson and Lipman, 1988] Pearson, W. and Lipman, D. (1988). Improved tools for biological sequence comparison. *Proc Natl Acad Sci*, 85:2444–2448.

[Pisanti et al., 2006] Pisanti, N., Carvalho, A., Marsan, L., and Sagot, M.-F. (2006). Risotto: Fast extraction of motifs with mismatches. In *Proceedings of the 7th Latin American Theoretical Informatics Symposium*, volume 3887.

[Pruesse et al., 2007] Pruesse, E., Quast, C., Knittel, K., Fuchs, B., Ludwig, W., Peplies, J., and Glöckner, F. (2007). SILVA: a comprehensive online resource for quality checked and aligned ribosomal RNA sequence data compatible with ARB. *Nucl Acids Res*, 35:7188–7196.

[Pruitt et al., 2009] Pruitt, K., Tatusova, T., Klimke, W., and Maglott, D. (2009). NCBI reference sequences: current status, policy and new initiatives. *Nucl Acids Res*, 37:D32–D36.

[Puglisi et al., 2007] Puglisi, S., Smyth, W., and Turpin, A. (2007). A taxonomy of suffix array construction algorithms. *ACM Computing Surveys*, 39:1–31.

[Qi et al., 2004] Qi, J., Luo, H., and Hao, B. (2004). CVTree: a phylogenetic tree reconstruction tool based on whole genomes. *Nucl Acids Res*, 1:W45–7.

[Qin et al., 2009] Qin, J., Li, R., Raes, J., Arumugam, M., Burgdorf, K. S., Manichanh, C., Nielsen, T., Pons, N., Levenez, F., Yamada, T., Mende, D., Li, J., Xu, J., Li, S., Li, D., Cao, J., Wang, B., Liang, H., Zheng, H., Xie, Y., Tap, J., Lepage, P., Bertalan, M., Batto, J.-M., Hansen, T., Le Paslier, D., Linneberg, A., Nielsen, H. B., Pelletier, E., Renault, P., Sicheritz-Ponten, T., Turner, K., Zhu, H., Yu, C., Li, S., Jian, M., Zhou, Y., Li, Y., Zhang, X., Li, S., Qin, N., Yang, H., Wang, J., Brunak, S., Dore, J., Guarner, F., Kristiansen, K., Pedersen, O., Parkhill, J., Weissenbach, J., Bork, P., Ehrlich, S. D., and Wang, J. (2009). A human gut microbial gene catalogue established by metagenomic sequencing. *Nature*, 464:59–65.

[Reis et al., 2012] Reis, D. C., Belazzougui, D., Botelho, F. C., and Ziviani, N. (2012). CMPH: C Minimal Perfect Hashing library.

[Richter et al., 2008] Richter, D. C., Ott, F., Auch, A. F., Schmid, R., and Huson, D. H. (2008). MetaSim–a sequencing simulator for genomics and metagenomics. *PLoS ONE*, 3(10):e3373.

[Roberts et al., 2013] Roberts, R. J., Carneiro, M. O., and Schatz, M. C. (2013). The advantages of SMRT sequencing. *Genome Biol*, 14:405.

[Rognes, 2011] Rognes, T. r. (2011). Faster Smith-Waterman database searches with intersequence SIMD parallelisation. *BMC Bioinformatics*, 12:221.

[Ronaghi et al., 1996] Ronaghi, M., Karamohamed, S., Pettersson, B., Uhlén, M., and Nyrén, P. (1996). Real-time DNA sequencing using detection of pyrophosphate release. *Anal Biochem*, 242:84–89.

[Ronaghi et al., 1998] Ronaghi, M., Uhlén, M., and Nyrén, P. (1998). A sequencing method based on real-time pyrophosphate. *Science*, 281:363–365.

[Ross et al., 2013] Ross, M., Russ, C., Costello, M., Hollinger, A., Lennon, N., Hegarty, R., Nusbaum, C., and Jaffe, D. (2013). Characterizing and measuring bias in sequence data. *Genome Biol*, 14:doi:10.1186/gb–2013–14–5–r51.

[Rothberg et al., 2011] Rothberg, J., Hinz, W., Rearick, T., Schultz, J., Mileski, W., Davey, M., Leamon, J., Johnson, K., Milgrew, M., Edwards, M., Hoon, J., Simons, J., Marran, D., Myers, J., Davidson, J., Branting, A., Nobile, J., Puc, B., Light, D., Clark, T., Huber, M., Branciforte, J., Stoner, I., Cawley, S., Lyons, M., Fu, Y., Homer, N., Sedova, M., Miao, X., Reed, B., Sabina, J., Feierstein, E., Schorn, M., Alanjary, M., Dimalanta, E., Dressman, D., Kasinskas, R., Sokolsky, T., Fidanza, J., Namsaraev, E., McKernan, K., Williams, A., Roth, G., and Bustillo, J. (2011). An integrated semiconductor device enabling non-optical genome sequencing. *Nature*, 475:348–352.

[Russell and Barton, 1992] Russell, R. and Barton, G. (1992). Multiple protein sequence alignment from tertiary structure comparison: assignment of global and residue confidence levels. *Proteins*, 14:309–323.

[Russo et al., 2009] Russo, L., Navarro, G., Oliveira, A., and Morales, P. (2009). Approximate string matching with compressed indexes. *Algorithms*, 2:1105–1136.

[Saitou and Ueda, 1994] Saitou, N. and Ueda, S. (1994). Evolutionary rates of insertion and deletion in noncoding nucleotide sequences of Primates. *Mol Biol and Evol*, 11:504–512.

[Salama and Stekel, 2013] Salama, R. and Stekel, D. (2013). A non-independent energy-based multiple sequence alignment improves prediction of transcription factor binding sites. *Bioinformatics*, page doi:10.1093/bioinformatics/btt463.

[Sanger et al., 1977] Sanger, F., Air, G., Barrell, B., Brown, N., Coulson, A., Fiddes, J., Hutchison, C., Slocombe, P., and Smith, M. (1977). Nucleotide sequence of bacteriophage ϕX174 DNA. *Nature*, 265:687–695.

[Schmieder et al., 2012] Schmieder, R., Lim, Y., and Edwards, R. (2012). Identification and removal of ribosomal RNA sequences from metatranscriptomes. *Bioinformatics*, 28:433–435.

[Schulz and Mihov, 2002] Schulz, K. and Mihov, S. (2002). Fast string correction with Levenshtein automata. *IJDAR*, 5:67–85.

[Segata and Huttenhower, 2011] Segata, N. and Huttenhower, C. (2011). Toward an efficient method of identifying core genes for evolutionary and functional microbial phylogenies. *PLoS ONE*, 6:doi:10.1371/journal.pone.0024704.

[Sheetlin et al., 2005] Sheetlin, S., Park, Y., and Spouge, J. L. (2005). The Gumbel pre-factor k for gapped local alignment can be estimated from simulations of global alignment. *Nucl Acids Res*, 33:4987–4994.

[Sinha and Zobel, 2004] Sinha, R. and Zobel, J. (2004). Cache-conscious sorting of large sets of strings with dynamic tries. *ACM JEA*, 9.

[Sinha et al., 2006] Sinha, R., Zobel, J., and Ring, D. (2006). Cache-efficient string sorting using copying. *ACM JEA*, 11.

[Smit et al., 2008] Smit, A., Hubley, R., and Green, P. (2008). Repeatmasker open-3.0. `http://www.repeatmasker.org`.

[Smith et al., 1986] Smith, L., Sanders, J., Kaiser, R., Hughes, P., Dodd, C., Connell, C., Heiner, C., Kent, S., and Hood, L. (1986). Fluorescence detection in automated DNA sequence analysis. *Nature*, 321:674–679.

[Smith and Waterman, 1981] Smith, T. and Waterman, M. (1981). Identification of common molecular subsequences. *J Mol Biol*, 147:195–197.

[Sorek and Cossart, 2010] Sorek, R. and Cossart, P. (2010). Prokaryotic transcriptomics: a new view on regulation, physiology and pathogenicity. *Nat Rev Genet*, 11:9–16.

[Staden, 1979] Staden, R. (1979). A strategy of DNA sequencing employing computer programs. *Nucl Acids Res*, 6.

[States et al., 1991] States, D., Gish, W., and Altschul, S. (1991). Improved sensitivity of nucleic acid database searches using application-specific scoring matrices. *Methods*, 3:66–70.

[Sved and Bird, 1990] Sved, J. and Bird, A. (1990). The expected equilibrium of the CpG dinucleotide in vertebrate genomes under a mutation model. *Proc Natl Acad Sci*, 87:4692–4696.

[Turnbaugh et al., 2007] Turnbaugh, P. J., Ley, R. E., Hamady, M., Fraser-Liggett, C. M., Knight, R., and Gordon, J. I. (2007). The human microbiome project. *Nature*, 449:804–810.

[Ukkonen, 1995] Ukkonen, E. (1995). On-line construction of suffix trees. *Algorithmica*, 14:249–260.

[Venter et al., 2001] Venter, J., Adams, M., Myers, E., Li, P., Mural, R., Sutton, G., Smith, H., Yandell, M., Evans, C., and Holt, R. (2001). The sequence of the human genome. *Science*, 291:1304–1351.

[Walser et al., 2008] Walser, J., Ponger, L., and Furano, A. (2008). CpG dinucleotides and the mutation rate of non-CpG DNA. *Genome Res*, 18:1403–1414.

[Wandelt et al., 2012] Wandelt, S., Rheinländer, A., Bux, M., Thalheim, L., Haldemann, B., and Leser, U. (2012). Data management challenges in next generation sequencing. *Datenbank-Spektrum*, 12:161–171.

[Wang and Samudrala, 2006] Wang, K. and Samudrala, R. (2006). Incorporating background frequency improves entropy-based residue conservation measures. *BMC Bioinformatics*, 7:385.

[Waterman and Vingron, 1994] Waterman, M. S. and Vingron, M. (1994). Rapid and accurate estimates of statistical significance for sequence data base searches. *Proc Natl Acad Sci*, 91:4625–4628.

[Weiner, 1973] Weiner, P. (1973). Linear pattern matching algorithms. *14th Annual IEEE Symp Switching & Automata Theory*, pages 1–11.

[Wilbur and Lipman, 1983] Wilbur, W. and Lipman, D. (1983). Rapid similarity searches of nucleic acid and protein data banks. *Proc Natl Acad Sci*, 80:726–730.

[Wommack et al., 2008] Wommack, K. E., Bhavsar, J., and Ravel, J. (2008). Metagenomics: read length matters. *Appl Environ Microbiol*, 74:1453–1463.

[Yan et al., 2013] Yan, Y., Xin, A., Zhu, G., Huang, H., Liu, Q., Shao, Z., Zang, Y., Chen, L., Sun, Y., and Gao, H. (2013). Complete genome sequence of a novel natural recombinant porcine reproductive and respiratory syndrome virus isolated from a pig farm in yunnan province, southwest china. *Genome Announc*, 1:doi: 10.1128/genomeA.00003–12.

[Yang and Yoder, 1999] Yang, Z. and Yoder, A. (1999). Estimation of the transition/transversion rate bias and species sampling. *J Mol Evol*, 48:274–283.

[Yi et al., 2011] Yi, H., Cho, Y., Won, S., Lee, J., Jin, Y., Kim, S., Schroth, G., Luo, S., and J., C. (2011). Duplex-specific nuclease efficiently removes rRNA for prokaryotic RNA-seq. *Nucl Acids Res*, 39:doi:10.1093/nar/gkr617.

[Zhao et al., 2012] Zhao, M., Lee, W. P., Garrison, E., and Marth, G. T. (2012). SSW library: An SIMD smith-waterman C/C++ library for use in genomic applications. http://arxiv.org/abs/1208.6350.

[Zhao and Boerwinkle, 2002] Zhao, Z. and Boerwinkle, E. (2002). Neighboring-nucleotide effects on single nucleotide polymorphisms: A study of 2.6 million polymorphisms across the human genome. *Genome Res*, 12:1679–1686.

List of Figures

List of Tables