INVITED PAPER

Short Read Mapping: An Algorithmic Tour

This paper discusses the challenge of mapping short DNA reads to an existing target genome, covering the approaches and the current tools for addressing this problem.

By Stefan Canzar and Steven L. Salzberg

ABSTRACT | Ultra-high-throughput next-generation sequencing (NGS) technology allows us to determine the sequence of nucleotides of many millions of DNA molecules in parallel. Accompanied by a dramatic reduction in cost since its introduction in 2004, NGS technology has provided a new way of addressing a wide range of biological and biomedical questions, from the study of human genetic disease to the analysis of gene expression, protein–DNA interactions, and patterns of DNA methylation. The data generated by NGS instruments comprise huge numbers of very short DNA sequences, or "reads," that carry little information by themselves. These reads therefore have to be pieced together by well-engineered algorithms to reconstruct biologically meaningful measurements, such as the level of expression of a gene. To solve this complex, high-dimensional puzzle, reads must be mapped back to a reference genome to determine their origin. Due to sequencing errors and to genuine differences between the reference genome and the individual being sequenced, this mapping process must be tolerant of mismatches, insertions, and deletions. Although optimal alignment algorithms to solve this problem have long been available, the practical requirements of aligning hundreds of millions of short reads to the 3-billion-base-pair-long human genome have stimulated the development of new, more efficient methods, which today are used routinely throughout the world for the analysis of NGS data.

KEYWORDS | Burrows–Wheeler transform; DNA sequencing; sequence alignment; string matching; suffix trees

I. INTRODUCTION

Sequence alignment has a long history in molecular biology and genetics, dating back to the first decoding of protein sequences in the 1960s. A pairwise alignment of two proteins identifies subsequences of amino acids that are position-wise similar, respecting the order in which they occur in the two proteins. The original goal of alignment was to uncover the evolutionary relationship between two proteins: positions that match represent the sequence of a common ancestor, and positions that differ represent mutations that occurred in one or both proteins since the divergence of their most recent common ancestor.

The first elegant algorithm to compute the optimal alignment of two sequences, in some sense the archetype for many alignment algorithms, was proposed by Needleman and Wunsch in 1970 [91]. It is based on the *dynamic programming* paradigm, which relates the optimal alignment of two sequences to the optimal alignment of two shorter substrings and orders the resulting subproblems in a way that avoids the repeated computation of identical nodes in the recursion tree. While such a *global alignment* is meaningful if two sequences are similar in their entirety, it might fail to discover local similarities. In 1981, Smith and Waterman [108] proposed an adaption of the Needleman–Wunsch algorithm that is able to find such *local alignments*, with a computational cost that is $O(nm)$, where n and m are the lengths of the two input sequences.

The biological and medical relevance of comparative sequence analysis emerged quickly as biological sequence

Manuscript received June 11, 2014; revised February 23, 2015; accepted July 6, 2015. Date of publication September 7, 2015; date of current version February 16, 2017. This work was supported in part by the U.S. National Institutes of Health under Grant R01 HG006102 to SLS.

S. Canzar is with the Center for Computational Biology, McKusick-Nathans Institute of Genetic Medicine, Johns Hopkins University School of Medicine, Baltimore, MD 21205 USA, and also with Toyota Technological Institute at Chicago, Chicago, IL 60637 USA (e-mail: canzar@ttic.edu).

S. L. Salzberg is with the Center for Computational Biology, McKusick-Nathans Institute of Genetic Medicine and the Institute of Genetic Medicine, Johns Hopkins University, Baltimore, MD 21205 USA, and also with the Department of Biostatics, Bloomberg School of Public Health, Johns Hopkins University, Baltimore, MD 21205 USA, and the Department of Computer Science, Johns Hopkins University, Baltimore, MD 21218 USA.

Digital Object Identifier: 10.1109/JPROC.2015.2455551

data became more widely available. This was exemplified by Doolittle's discovery [26] that the *sis* oncogene, a gene from a simian sarcoma virus, was highly similar to a previously known gene, platelet-derived growth factor (PDGF), which is involved in controlling cell growth. This very early alignment result, in 1983, led to the realization that oncogenes might be abnormally expressed versions of normal growth factors.

Sequence databases grew rapidly in the 1980s (GenBank[1] was started in 1982), and as a result, performing a full dynamic programming comparison of a query sequence to every known sequence soon became computationally very costly. The need to efficiently align a query sequence against a database motivated a heuristic algorithm proposed by Wilbur and Lipman [116] and refined and implemented in the FASTA program suite [93]. The basic principle underlying this algorithm is to *exclude* large parts of the database from the expensive dynamic programming comparison by rapidly identifying candidate sequences that share short stretches (k-tuples) of highly similar sequence with the query. FASTA was followed by the BLAST program [7], which achieved additional speed advantages and added a very useful feature: an estimate of the statistical likelihood that each matching sequence had been found by chance. BLAST soon became the most widely used search program for biological sequence databases, and the two primary publications (Altschul *et al.* 1990 [7] and 1997 [8]) now have over 90 000 citations. BLAST's combination of speed and sensitivity has made it the workhorse of biological sequence searches for over twenty years.

As whole-genome sequencing accelerated in the late 1990s and early 2000s, other alignment challenges emerged. For example, microbial genomicists began sequencing multiple strains of the same bacteria, motivating the need to align two complete genomes to one another. This requirement, which later included far larger insect, plant, and animal genomes, presented a requirement that BLAST did not address. The first solution to this problem, the MUMmer program [24], used the suffix tree data structure, which had not previously been used for DNA sequence alignment. Suffix trees have the advantage that they only occupy linear space (in terms of the length of the input sequences) and they can be searched for exact matches in linear time.

Other methods for fast indexing also emerged at this time, such as the BLAT [49] and SSAHA [92] programs, both of which (similarly to FASTA) used a hash index based on k-tuples. These methods focused on hashing into vertebrate genomes (especially human) using an index that could fit into RAM and therefore allowed much faster searching than BLAST, although neither provided probability estimates in their output.

The advent of ultra-high-throughput next-generation sequencing (NGS) technology in 2007 presented major new challenges for alignment. The first sequencers from Solexa (later Illumina) were able to generate, in a single run, tens of millions of very short DNA sequences, or "reads." Initially, read lengths were only 25 base pairs (bp), which although short is nonetheless long enough to map approximately 79% of the human genome [68]; i.e., 79% of the genome is composed of 25-bp sequences that occur only once. Because 25-bp reads can be aligned uniquely to the genome, this technology gained immediate use as a new method for studying human genetic variation and disease. Read lengths subsequently increased, and have now reached 150 bp in the highest throughput sequencing machines.

The rapid improvement in sequencing capacity has dramatically outpaced Moore's Law, according to which computing speed doubles every two years. Today, an Illumina HiSeq 2500 in rapid mode can produce 120 gigabases, the equivalent of 40 human genomes, in just 27 h,[2] representing a speedup of approximately 500 000-fold since 2001. This ultra-high-throughput technology and the accompanied dramatic reduction in cost have opened the door to a remarkable variety of biological and biomedical applications. Besides determining the complete genomes of numerous species [89], it allows the resequencing of large cohorts of individuals and has spurred efforts to sequence thousands of individual humans [1]. Illumina's recent HiSeq X Ten platform can sequence more than 18 000 human genomes per year at a marginal cost of just $1000 per genome.[3] Additional new techniques have 2511led to the widespread use of sequencing to study gene expression (RNA-seq [21], [72], [87]), protein–DNA interactions (ChIP-seq [45]), and DNA methylation (MeDIP-seq [27]).

These advances in sequencing throughput have presented a major challenge for alignment algorithms. Because CPU speeds have been largely constant, the time required merely to align reads to the genome has increased enormously. This in turn has led to new efforts to improve alignment algorithms. In this perspective, we describe these efforts and the current state of the art in NGS alignment algorithms.

Since 2007, computational biologists have developed more than 70 read mapping tools [35] that try to tackle these challenges. This survey elucidates the algorithmic underpinnings that paved the way to extremely efficient read mapping software, which today is used throughout the world for next-generation sequence alignment. We do not attempt to evaluate the mapping tools themselves, but rather to describe their main algorithmic strategies and their strategies for making speed versus accuracy tradeoffs from an *algorithm engineering* point of view.

[1]http://www.ncbi.nlm.nih.gov/genbank/

[2]http://www.illumina.com/systems/hiseq_2500_1500/performance_specifications.ilmn

[3]http://www.illumina.com/systems/hiseq-x-sequencing-system/system.ilmn

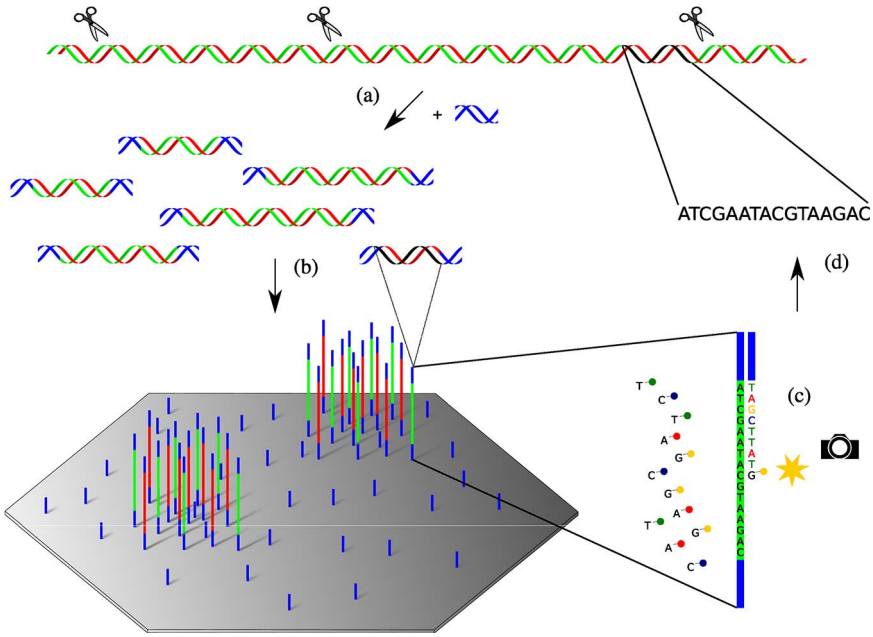

Fig. 1. *Workflow of next-generation sequencing. (a) Genomic DNA is sheared into fragments, and platform-specific adapter sequences (blue) are then attached. After amplification on a solid surface (b), the sequence of nucleotides is "read out" (c) from signals emitted when a base is added to the complement of a template strand. A read mapping algorithm then must find (d) the genomic origin of the resulting reads.*

A. Short Read Mapping Problem

1) Data: Each next-generation sequencing platform implements a different technology, but the workflow can be illustrated by a brief summary of the Illumina method (see Fig. 1). First, genomic DNA is sheared into *fragments* that can be size-selected to obtain a template library of uniformly short DNA fragments. Platform-specific adapter sequences are then attached to both ends of the fragments, which allow them to be attached to a solid surface. Through a local PCR amplification step, many copies of each template molecule are produced in a tightly clustered location on the surface. These molecules emit a signal during the sequencing reaction that is strong enough to be detected by the optical system of the instrument. During sequencing, every molecule is cyclically extended by a single base, and all clusters are read in parallel before the cycle is repeated. A *base-calling* algorithm determines individual bases from raw signal data, to which quality scores are assigned. Since the precise meaning and range of quality scores is platform dependent, here we assume that a higher quality score indicates a lower likelihood of the base call being incorrect. In *paired-end* sequencing, both ends of the original DNA fragment are sequenced, and the instrument keeps track of this pairing information.

2) Solving the Inverse Problem: The first step in the analysis of NGS data is to determine where in the genome each of the short reads originated [Fig. 1, step (d)]. The obvious criteria for mapping such a read to a genomic location is sequence similarity. Due to sequencing errors and genuine differences between the reference genome and the sequenced organism, a read might not match its corresponding location in the reference genome exactly. We therefore need an alignment method that permits some number of mismatches, insertions, and deletions. Although older alignment software such as BLAST [7] or BLAT [49] can be used to map short reads to a reference genome, those methods are simply too slow. The sheer amount of data, sometimes comprising billions of short reads that have to be aligned to a large (e.g., mammalian) genome, has required innovative new algorithms that run orders of magnitude faster, at least for the specialized problem of short-read alignment to a fixed reference genome. In order to be most useful, these algorithms should also exploit additional information associated with the experimental protocol, such as read pairing information, and take into account technology-specific error profiles.

In the case of paired-end reads, we expect the two *mates* to map at a distance and orientation that is consistent with the library fragment length and the relative sequencing direction, respectively. In the formalization of the read mapping problem below, however, we ignore pairing information on reads and treat the mates as independent single-end reads. In Section VII-A we describe strategies to incorporate the additional constraints imposed by paired-end information.

Because the DNA strand from which a read originates is unknown, either the read sequence itself or its reverse complement may be mapped to the reference genome. In

the problem formulation below and in our description of the algorithms, we will not make a distinction between a read and its reverse complement and simply refer to them both as *reads*.

B. Read Mapping as String Matching

Given a reference genome G and a set of reads $\mathcal{R}$, in the *read mapping* problem we want to determine for every read $R \in \mathcal{R}$ its origin in G. The reference G and the reads in $\mathcal{R}$ can be modeled as *strings*, (mathematical) sequences of symbols from alphabet $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$. The symbols in Σ represent the four nucleotides *adenine*, *cytosine*, *guanine*, and *thymine*. To conform to biological usage, we use the terms *string* and *sequence* synonymously. In contrast to a *subsequence* of a string S, however, a *substring* denotes a *contiguous* subsequence of S.

1) Exact String Matching: The origin of a read R might be detected through *sequence identity* of R to a substring in G. Because multiple occurrences of R in G cannot be distinguished without incorporating additional information, we are interested in finding *all* occurrences of R in G.

Definition 1: Let Σ be a finite alphabet of size $|\Sigma| = \sigma$. Given a *text* $T \in \Sigma^*$ of length $|T| = n$ and a *pattern* $P \in \Sigma^*$ of length $|P| = m$, the *exact string matching* problem is to find all occurrences of P in T.

Here, the pattern P corresponds to a single read in $\mathcal{R}$, the reference genome is represented by T, and $\sigma = 4$. Since $\mathcal{R}$ typically comprises tens to hundreds of millions of reads and $|T|$ is huge compared to the read length, classical matching algorithms like Boyer–Moore [13] and Knuth–Morris–Pratt [52] are impractical for the read mapping problem. For every pattern P, these algorithms require time proportional to $|T|$ to find occurrences of P in T.

Indexed string matching builds an auxiliary data structure (an index) for T such that occurrences of pattern P can be found efficiently without scanning the complete text T. Such an index structure is indispensible in the context of mapping short reads to a genome because it speeds up the mapping of many tens of millions of reads and thereby marginalizes the cost of constructing the index. In addition, the reference genome (T) is static, and thus the cost of maintaining the index is low.

Suffix trees [9] and *suffix arrays* [83] are classical *full-text* index structures, the latter being, as an array of integers specifying the lexicographical order of the suffixes of T, a space efficient alternative to suffix trees. They permit (almost) optimal $\mathcal{O}(m+occ)$ and $\mathcal{O}(m+\log n+occ)$ search time, respectively, where occ is the number of occurrences of P in T. Because search time is a function of the length of the pattern P rather than the genome, these represent a much faster way to map short reads. The initial barrier to their widespread use in computational biology was their space requirement. Both data structures require $\Theta(n \log n)$ bits and are thus asymptotically larger then the text itself, which needs $n\lceil \log \sigma \rceil$ bits. With $\sigma = 4$ and $n > 3 \cdot 10^9$ for the human genome, the resulting demand in memory resources can be prohibitively large, in particular when the index is to be stored in main memory to allow fast access.

Although several attempts were made to reduce the space requirement of index structures while preserving fast search functionality (e.g., [48]), it was not until Ferragina and Manzini developed their full-text index [32], based on the Burrows–Wheeler transform, that anyone had an index that was comparable in size to the text itself. They introduced the first *self-index*, which takes space proportional to the *compressed* text, provides fast searching, and contains sufficient information to reproduce the text and thus can replace the text. In essence, the proposed *FM-index* requires space close to the desired entropy bound for constant-sized alphabets and allows text search with almost optimal time complexity. Theoretical concepts underlying this exciting invention and its further developments are surveyed in [90], their engineering aspects are addressed in [31].

2) Approximate String Matching: Alignment methods that only allow exact matches of reads to the genome are, for all practical purposes, useless. Real data have two significant sources of mismatches: sequencing errors and true variation. Sequencing errors vary with the technology of sequencing; the current error rate of high-throughput Illumina sequencers is less than 0.5%. True variation refers to differences between any human subject (if we are aligning human sequence) and the reference genome. In the human population, the overall variation has been estimated at approximately 0.1% (1 base per 1000), but this varies greatly depending on the background: The reference human genome is a European male, and the genomes of individuals with different ancestry have much higher rates of variation. To be useful, aligners must be able to tolerate at least as much variation as is found in the population.

More formally, any practically useful algorithm must be able to find all approximate occurrences of a read; i.e., substrings of T whose similarity to P exceeds a specific threshold, or whose distance is less than a similar threshold. The two most commonly used distance functions are *Hamming distance* [98] and *Levenshtein* or *edit distance* [61]. While Hamming distance simply counts the number of substitutions implied by mismatched symbols of two strings at the same position, edit distance additionally accounts for inserted and deleted symbols (*indels*) in one string with respect to the other.

Definition 2: Given text T and pattern P, the *k-mismatch problem* asks for all $|P|$-length substrings of T within Hamming distance k, i.e., that match at least $|P| - k$ characters in P.

In contrast, a similarity measure is based on a scoring scheme that assigns a higher score to matched symbols

```
AACTAGA-AC-TACTGA
AA-TACAGACTTAC-GA
```

Fig. 2. *Alignment of two strings, implying one substitution (red) and four insertions/deletions (gray).*

than to mismatched symbols, insertions, and deletions, and in the context of short read mapping ideally incorporates base quality scores. A set of insertions and deletions implying matches and mismatches between two sequences is represented by an *alignment* (Fig. 2). A (global) alignment of two DNA sequences inserts a dash symbol "–" $\notin \Sigma$ into either string to obtain two strings of the same length. This alignment establishes a positionwise correspondence between nucleotides in the augmented strings. Nucleotides aligned to dash symbols represent either an insertion in one string or a deletion in the other. The *score* or *value* of an alignment is the sum of its position-wise scores.

The (weighted) edit distance or *similarity* (score) between two strings is defined by an optimal alignment that minimizes the number (or total weight) of edits (substitutions, insertions, deletions) or that has a maximum score, respectively. Assigning a score of 1 to matching nucleotides, -3 to mismatches, and -2 to indels, the alignment in Fig. 2 is optimal and implies a similarity score of 1. The (unit cost) edit distance of the two sequences is 5.

Definition 3: Given text T and pattern P, the *k-errors problem* asks for all substrings of T within edit distance k.

Algorithms that compute an optimal alignment for two sequences of length n and m, based on the original Smith–Waterman [108] paradigm, run in time $\mathcal{O}(nm)$ for typical scoring schemes. Variants that simply bound the number of errors k (Defintions 2 and 3) allow for a faster $\mathcal{O}(kn)$ solution.

The deletion or insertion of several consecutive nucleotides by a single mutational event can by captured by the concept of a *gap*. A gap is a maximal sequence of dash symbols in one string, which is penalized by a weight that is a function of the length of the gap. In the context of short read mapping, the most commonly used model gap penalties are *affine* functions of the form $\alpha + q\beta$, where q denotes the length of the gap. In the case of affine gap weights, the optimal alignment can be found within the same time bound of $\mathcal{O}(nm)$ as for scoring schemes without gaps.

Finding *all* approximate occurrences of a short read in a reference genome requires a modification of the base condition in the dynamic programming algorithm to allow the read to start at any position in the genome. Although this will solve the alignment problem, it is clearly infeasible to spend $\mathcal{O}(nm)$ time, or even $\mathcal{O}(kn)$ in case of an error bound k, to match each of hundreds of millions of reads approximately to a large reference genome.

C. Overview

The static nature of most genomes (human, mouse, and other reference genome assemblies) allows the use of a precomputed index to speed the search process. Indexing a text to support approximate matching queries is a challenging problem, primarily because of their space requirements. Classical full-text indexes such as suffix trees and suffix arrays require, in the most optimized implementations, approximately 3 B per base or 36 GB of memory for a mammalian genome.

In Section II, we present a class of methods that attempt to quickly eliminate large parts of the genome where the read does *not* match. *Filtering algorithms* typically rely on unaltered parts of the read that do not contain any substitutions or indels and reduce the search for inexact-matching regions to the exact matching problem, which can be solved efficiently through a lookup in a hash table. Only regions that match a sufficiently large part of the read exactly must be verified by a more expensive approximate matching algorithm.

In Section III, we describe how state-of-the-art software tools apply the FM-index to map short reads to a reference genome. The FM-index has originally been designed for the exact string matching problem and as such supports filtration-based approaches. Furthermore, it allows to navigate through occurrences of related patterns and thus facilitates an inexact search of the read.

Section V discusses adaptations of algorithms and data structures that are necessary to support the massive parallelism provided by a graphics processing unit (GPU). Fig. 16 organizes methods described in Sections II–V in a tree of algorithmic design decisions.

Section VI summarizes efforts made by read mappers to identify the true origin of a read among its approximate occurrences and to provide an estimate on the probability that the true origin was found.

In Section VII, we address aspects of the read mapping problem that are not captured by an approximate string matching model. We conclude in Section VIII.

II. HASHING-BASED METHODS

The alignment challenges posed by NGS sequencing technology were initially addressed by methods that used the seed-and-extend strategy (e.g., [8] and [81]). This approach assumes that in the true alignment of a read, parts of the read, the *seeds*, match the reference without errors. If these seeds are long enough to match in only a small number of places, the aligner can collect all the seed matches and then extend each of them. Seeding quickly discards large parts of the genome where the pattern does *not* match according to a given error model. For example, if neither `ACA` nor `AGT` occur in a given text, then `ACAAGT` with at most one mismatch cannot exist either. Only regions around the seeds need to be evaluated with a more costly approximate string matching algorithm.

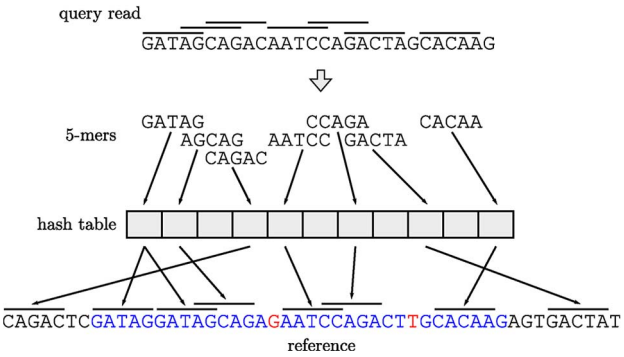

Fig. 3. *Workflow of hashing-based methods. First, k-mers are extracted from the query read applying seed templates. Here, k-mers correspond to substrings of the read, indicated by black lines. Typically the seed template is applied at all positions of the read. A hash table returns the position of k-mer occurrences in the genome. In the example shown, an approximate match (blue) of the query read with two mismatches (red) is hit by six seeds.*

Assuming that seeds have length k, we can locate k-mers shared between the reads and the genome through a hash table index. Aligners either build a hash table of subsequences found in the genome and scan the reads for matching k-mers, or the other way around. The hash table is accessed through a k-mer and stores the positions where the corresponding subsequence occurs in the genome or the read. Tools that index the reads include Eland (AJ Cox, Illumina, unpublished), mrFAST/mrsFAST [6]/[39], MAQ [64], RMAP [106], ZOOM [70], SeqMap [44], and SHRiMP [97]. Tools that build a hash index of the genome included SOAP [66], Novoalign,[4] Mosaik [60], SRmapper [36], Stampy [79], BFAST [42], SHRiMP2 [22], Hobbes [3], and FastHASH [118]. For a more comprehensive list of sequence aligners, see the "Short-Read Sequence Alignment" section at http://en.wikipedia.org/wiki/List_of_sequence_ alignment_software and http://wwwdev.ebi.ac.uk/fg/hts_mappers/.

Fig. 3 depicts a typical workflow for a hash-based mapper. First, it extracts k-mers from a query read by applying the same seed template as was used to index the reference genome. In the example shown in Fig. 3, the 5-mers are obtained from (contiguous) substrings of length 5. Some methods, including BLAST and Subread [69], only consider an informative subset of the resulting k-mers in subsequent steps. Second, the hash table is queried to locate all occurrences of the query k-mers in the reference genome. Third, genomic regions containing single or multiple seed matches are aligned using an approximate alignment method that attempts to extend the seeds to a full read alignment, using a scoring function that is particular to each mapper.

Next, we discuss different seeding strategies to select k-mers that have to be matched between read and reference genome. We give examples of additional filters that can further reduce the number of candidate regions to be verified in Section II-B. Algorithms employed during the final verification step depend on the error model and are reviewed in Section II-C.

[4]http://www.novocraft.com

A. Seed Template Design

The "guessing" of an exact matching seed is usually guided by a filtering criterion that relies on parts of the read that must remain unaffected by the edit operations allowed by a specific error model. Seed templates specify these potentially unaffected parts (subsequences) of the reads and are important for the efficiency of filtering-based methods. A low *selectivity* gives rise to a large number of seeds that have to be extended by computationally intensive dynamic programming algorithms (see Section II-C). A low *sensitivity* will result in many approximate matches that are missed by the filter, which might include the true origin of a read. In contrast, a *lossless* filter is guaranteed to not discard any true occurrence.

1) Lossy Filtration: In the simplest case, as typified by the BLAST program [7], the algorithm creates seeds from consecutive sequences of characters that must occur in both the read and the genome, similar to the six matching seeds in Fig. 3. Requiring a long sequence to match exactly might cause the correct alignment to be missed, while a short sequence may have an excessive number of false hits that will not extend to full alignments. The highly repetitive structure of many genomes may greatly increase the cost of overly short seeds, which can match in literally millions of locations on large plant and animal genomes.

In PatternHunter [81], Ma *et al.* addressed this dilemma by introducing *spaced seeds* that require a *non*consecutive sequence of characters to match. A spaced seed *template* can be represented by fixed-length words over the alphabet {0, 1}, where 1's indicate positions that are required to match and 0-positions are ignored. Fig. 4 depicts a seed match implied by template 1101011. In this model, consecutive templates as used by BLAST correspond to templates of the form 1^k.

If appropriately designed, a spaced seed has a higher chance of hitting a true approximate match on the genome while producing fewer random hits. Besides spanning a larger interval on the read with the same number of positions required to match, the main property responsible for

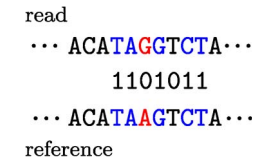

Fig. 4. *5-mer* TAGCT *matched by the seed template 1101011 is effectively* TANGNCT*, where* N*'s correspond to "don't care" positions. Mismatches (red) at these positions are ignored.*

this advantageous behavior is the increased independence of a spaced seed template and its shifted copies. Simply speaking, a spaced seed template and its shifted instance share fewer positions that are required to match and thus together capture a wider range of mismatch scenarios. Notice that spaced seeds seek for parts of the read that match with small Hamming distance, but do not account for insertions and deletions.

Again, increasing the selectivity of a spaced seed by requiring a higher number of positions to match usually comes at the cost of a lower sensitivity. An alternative scheme was proposed in [109] and implemented in PatternHunter II [65]. It uses *multiple* (spaced) seeds in *disjunctive* form, that is, a seed must match under at least one template. Intuitively, the loss in sensitivity resulting from an increased number of positions required to match is compensated by alternative templates. On the downside, multiple seeds multiply the computational effort necessary to detect matching seeds. BFAST [42], for example, aims at determining a small set of seed templates by greedily adding a seed template that maximizes the gain in sensitivity.

SEME [20], on the other hand, is based on a single consecutive seed template (similar to BLAST), which is however not simply shifted along the read but skips a certain number of bases. Intuitively, the idea is similar to the one underlying spaced seeds: Two consecutive seeds with less overlap capture a wider range of error scenarios. The sequential seed mapping strategy of SEME, which stops as soon as the first hit is encountered, leverages this effect. SEME does not use a hash index to map the seeds but finds (exact) occurrences on the genome through a tailored binary search in a sorted list of integers representing 32-mers from the genome.

The methods discussed so far identify candidate mapping locations based on individual seed matches between read and genome, that is, seeds are treated *disjunctively*. In contrast, the *seed-and-vote* scheme proposed in [69] relies on multiple seeds that *jointly* define a candidate region. The idea is to use the *number* of exactly matching (consecutive) seeds as a proxy for the edit distance. Using the expectation that a small edit distance should yield more contiguous stretches of exact matches in the corresponding alignment, the Subread program [69] allows all the seeds of a read to vote for the best mapping location. The region receiving the highest number of votes is identified as the (one) final mapping location, which is not guaranteed to lie within a certain error threshold.

Similarly, NextGenMap [103] and SHRiMP [97] consider candidate regions that are supported by a sufficiently large number of seed matches. The corresponding threshold is inferred in NextGenMap from the distribution of the number of mutually consistent seed matches. Simply speaking, reads from repetitive regions necessitate the verification of a higher number of candidates compared to reads from unique regions.

2) Lossless Filtration: Lossless filtration algorithms attempt to remove potential regions from consideration without losing any true matches. ZOOM [70], for instance, designs minimum cardinality sets of spaced seeds needed to achieve full sensitivity for a given read length, allowed number of mismatches, and number of positions required to match. Most of the methods in this section, however, either apply the pigeonhole principle or rely on a lower bound on the number of required seed matches. While the former naturally applies to the k-mismatch problem, the lower bound criterion is used to identify candidate regions for k-error matches.

a) Seed design by the pigeonhole principle: Methods that are guaranteed to detect all k-mismatch occurrences of a read often employ a combinatorial argument related to the pigeonhole principle to design (multiple) spaced seed templates. For example, Eland, MAQ, SeqMap, and SOAP subdivide the read into four parts of roughly the same length, such that two mismatches are guaranteed to leave two parts unaffected. MAQ applies this strategy only to the first, most reliable, 28 bases of the read. Six (spaced) seed templates cover all possible combinations of two error-free parts. For 8 bases, for example, the templates are

11110000	00111100
00001111	11001100
11000011	00110011.

Alternatively, RMAP originally used the Baeza–Yates and Perleberg filtration method [10] and partitioned the read into $k+1$ contiguous templates, such that an approximate match with at most k mismatches is guaranteed to contain one error-free part. A similar partitioning strategy is applied in one of the two filters implemented in RazerS 3 [115]. On the negative side, a higher number k of allowed mismatches shrinks the minimum length $\lfloor m/(k+1) \rfloor$ of the contigous templates required to achieve 100% sensitivity, yielding a higher number of false positive hits that have to be evaluated subsequently. A more selective (and relatively large) set of spaced seed templates is employed by an updated version of RMAP, which obtains full sensitivity to two mismatches in the first 32 bases of the read.

If the read is partitioned into $s < k+1$ nonoverlapping seeds, their increased length comes at the cost of losing the guarantee that an error-free seed exists. Nevertheless, by the (generalized) pigeonhole principle each approximate occurrence of the read with at most k errors contains a seed with at most $\lfloor k/s \rfloor$ errors. Such an *approximate occurrence* can be found by *backtracking* search in the Masai program [105] (see Section III-E).

In contrast, Hobbes2 [50] selects $k+2$ nonoverlapping seeds and infers candidate regions from two or more seed matches. A bounded number of indels are accounted for by

allowing gaps between two matched seeds and by expanding the candidate region.

Because the repetitiveness of equal-length seeds can vary considerably for a given read, the GEM mapper [84] and Hobbes(2) [3], [50] determine the partitioning into seeds adaptively, keeping the number of matches of each seed or the total number of matches small, respectively. From the resulting number of seeds, GEM infers the maximal number of errors that has to be allowed to guarantee full-sensitivity (see Section III-E).

b) Lower bound on seed matches: Following a similar intuition as the seed-and-vote paradigm (Section II-A1) more rigorously, Rasmussen *et al.* [95] proposed a filter that is based on multiple seeds that *conjunctively* indicate a candidate mapping location. It is based on the observation that a pattern P and an approximate occurrence of P in the text (genome) with (Levenshtein or Hamming) distance k have at least $|P| + 1 - q(k+1)$ substrings of length q (q-grams) in common (q-gram lemma) [47]. This bound is optimal. Intuitively, each of the k errors affects at most q of the $|P| - q + 1$ many q-grams of P. For example, the single mismatch between strings in Fig. 5 affects the three 3-grams marked by the red lines and leaves $8 + 1 - 3 \cdot 2 = 3$ 3-grams indicated by blue lines unchanged.

The q-gram lemma was first used in the database searching algorithm QUASAR [15] and was later generalized to gapped q-grams [16] and families of gapped q-grams [54].

In contrast to the spaced seeds discussed above, a filter criterion that builds upon this lower bound on the number of q-grams takes into account insertions and deletions by which the read may differ from its approximate match. Relying on multiple seeds that flag a candidate region of approximate occurrence is a technique commonly applied in long-read alignment; e.g., see BLAT [49], AGILE [86] and Section III-F.

Rasmussen *et al.* [95] show that the search for candidate regions sharing a sufficient number of q-grams (the q-gram lemma) can be restricted to specific parallelograms in the dot plot of the sequences (Fig. 6). After building an index containing the locations of q-grams, these parallelograms can be found using a sliding window technique. If the number of diagonals in the edit matrix that are summarized into bins is chosen appropriately, the update of the number of common q-grams falling into certain bins as the window proceeds can be performed using fast bit-operations. Such a scheme is implemented, for example, in RazerS [114] and its successor RazerS 3 [115].

Fig. 5. ***Single mismatch does not affect three q-grams (blue lines), but invalidates three seed matches (red lines).***

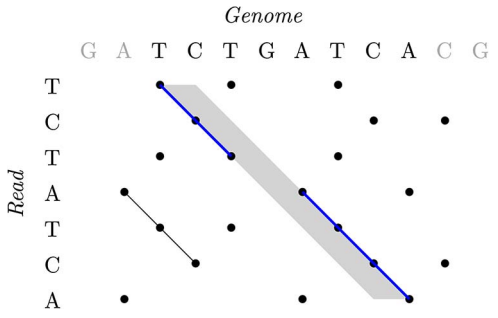

Fig. 6. ***Dot plot between read*** `TCTATCA` ***and its approximate occurrence of Levenshtein distance 1 in the genome. Dots mark identical nucleotides in the two sequences written along the vertical and horizontal axes. The parallelogram (gray) spans 8 columns and 2 diagonals and contains three 3-hits (blue).***

FastHASH [118], on the other hand, uses a lower bound on the number of consecutive seeds that are required to satisfy an adjacency condition. Seeds that are adjacent in the read are required to match to adjacent positions on the reference, where the adjacency condition is slightly relaxed if indels are allowed. The minimum number of adjacent seed matches that define a candidate region is determined by the maximal number of errors allowed and the number of seeds into which the read is divided (using the pigeonhole principle). Compared to the set of candidate regions implied by independent seed matches, the adjacency requirement eliminates false positive hits only and thus achieves full sensitivity with respect to a given edit distance threshold. To reduce in turn the comutational cost of the adjacency filter, which involves a binary search on the list of seed locations on the reference, FastHASH chooses "cheap" seeds in increasing order of the number of matches on the reference genome. The authors of FastHASH argue that this simple sorting scheme exhibits a better cost-efficiency tradeoff than the dynamic program applied by Hobbes (Section II-A2a) to determine an optimal division of reads. The latter involves a significant number of hash-table accesses which potentially outweigh the reduction in k-mer query costs.

B. Additional Filters

Since the verification of a large number of candidate regions by a dynamic programming scheme (Section II-C) can be computationally expensive, GASSST [98] and Hobbes [3] apply additional filters to further eliminate false positive seed hits prior to their extension. Two types of filters in GASSST provide lower bounds on the edit distance between a read and a potential mapping location on the reference. If the lower bounds exceed a given error threshold the seed hit is discarded. The *frequency vector filter*, introduced in [112], eliminates seed hits that, together with flanking regions, exhibit too much deviation in nucleotide frequency from the genomic region. A similar

filter is applied by Hobbes [3]. Second, GASSST utilizes, similarly to the PASS aligner [18], precomputed alignment scores of all possible words of a given (short) length to derive a lower bound on the alignment score of a seed and its flanking regions. To support this filtering mechanism, the hash table additionally stores flanking regions of k-mers. Notice however that these filters are based on a simple, unweighted edit distance. Hobbes additionally allows one to compute a lower bound on the number of mismatching characters (Hamming distance) through bitwise operations.

C. Seed Extension

Candidate regions returned by, for example, a pigeonhole filter or a q-gram based filter (Section II-A2) and that have optionally passed additional filters (Section II-D) have to be verified to contain an approximate occurrence of the read according to a specified error model. More specific distance measures usually allow faster algorithms to be applied.

1) Hamming-Based Extension: If an approximate match is defined with respect to a maximal number of mismatches (k-mismatch problem), a genomic region can be verified simply through a linear scan counting the number of mismatches, as is done, e.g., in RazerS and RazerS 3. At essentially no additional cost, MAQ minimizes the sum of quality scores of mismatched bases to obtain "more reliable" alignments. Alternatively, bitwise operations (mainly XORs) between an appropriate binary representation of read and genomic region can be employed to obtain a bit vector that indicates mismatches. The number of 1 bits in this vector and thus the number of mismatches can be determined through a look-up table (e.g., SOAP) or using a technique proposed by Warren [113] (e.g., ZOOM, RMAP). This bit-parallel scheme does not directly account for quality scores. RMAP incorporates quality values at a lower resolution by treating low-quality bases, defined in terms of a threshold, as wildcards always inducing a match.

2) Alignment-Based Extension: A very restricted edit distance is considered in SEME [20]. Relying on a low indel error rate of the Illumina sequencing platform and a small number of indel polymorphisms between reference and sequenced subject, it allows only one insertion or deletion. The proposed extension algorithm then runs in time linear in the length of the read.

Generally, if the approximate matching criteria is based on a similarity score, the Smith–Waterman algorithm is the method of choice to compute a (gapped) local alignment (see, e.g., BFAST). The gaps between mapped seeds in the seed-and-vote scheme [69] (Section II-A1) are filled by a Smith–Waterman dynamic program that takes into account the indel length that is inferred from distance discrepencies of seeds between read and genome.

Several schemes have been proposed that build on Single-Instruction Multiple-Date (SIMD) instructions available on modern CPUs to compute several cells of the dynamic programming matrix in parallel [30]. Stampy, Novoalign, and SHRiMP are example methods that employ SIMD-vectorized Smith–Waterman implementations. SHRiMP devises a new scheme that increases the degree of parallelism at the cost of supporting only uniform match and mismatch scores.

Even faster bit-parallel algorithms exist if quality scores are neglected and the mere number of edit operations (k-errors problem) is used as a measure of relatedness. RazerS, RazerS 3, Masai [105], Hobbes [3], and GEM [84] implement (a banded variant [43] of) Myers bit-vector algorithm [88].

III. BWT-BASED METHODS

Among the most widely used software tools used for mapping short DNA reads to a reference genome are Bowtie [59] and its successor Bowtie 2 [57], as well as BWA [62] and BWA-SW [65]. At the core of all of these programs lies the FM-index [33], which allows them to use the *Burrows–Wheeler transform* (BWT) [17] as a suffix array-like index structure. Simply speaking, the FM index augments the space-efficient BWT with additional data that permit very fast exact string matching. In Section III-A and B, we briefly summarize key properties of the BWT and the main algorithmic ingredients that make the BWT searchable at the cost of a small amount of additional memory.

Prior to their use in NGS alignment, BWT-based indexes such as the FM index and the *compressed suffix array* [38] had been used for several other problems in computational biology, including repeat finding [41], whole-genome comparison [71], the design of whole-genome tiling arrays [37], [94], the local alignment of a long pattern (e.g., a gene) to a reference genome [58], and more recently, it has also been employed in genome assembly [104]. Although the problem studied in [56] is mathematically very similar to the short read alignment problem, the exact method the authors proposed does not provide a practical solution to the alignment of hundreds of millions of short reads. In Sections III-C to E, we show how state-of-the-art mapping tools modify and extend the BWT and the FM index so they can handle inexact matching as well as many specific characteristics of next-generation sequencing reads.

A. Burrows–Wheeler Transform

The BWT T^{bwt} is a reversible permutation of a string T that facilitates compression through its repetitive structure, and that has proved useful in lossless data compression techniques such as bzip2. It is constructed by: 1) appending character $\$ \notin \Sigma$ to T; 2) creating a conceptual matrix $\mathcal{M}_T$ (the Burrows–Wheeler matrix) whose rows are the cyclic rotations of $T\$$ in lexicographical order; and finally 3) extracting the last column of $\mathcal{M}_T$. Fig. 7 illustrates steps 1)–3) for an example DNA sequence. Sorting the rows of $\mathcal{M}_T$ is

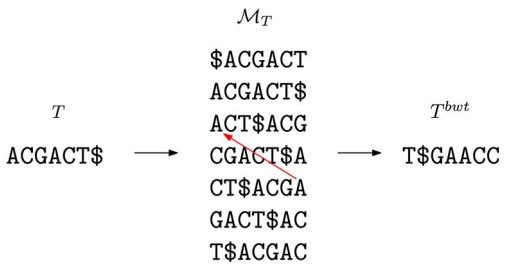

Fig. 7. *Constructing the BWT of* ACGACT*. LF mapping (red arrow): The second occurrence of* A *in the last column corresponds to the second occurrence of* A *in the first column.*

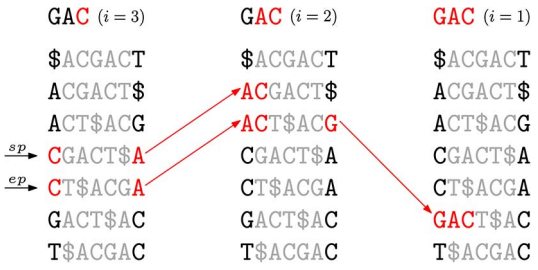

Fig. 8. *Backward search of* $P =$ GAC *in BWT of* $T =$ ACGACT*. The red arrows denote the update of sp and ep in lines 3 and 4 of Algorithm 1.*

equivalent to sorting the suffixes of T, which shows the close relationship between matrix $\mathcal{M}_T$ and the suffix array built on T.

A key feature of the Burrows–Wheeler Matrix $\mathcal{M}_T$ is the *Last-to-First column mapping LF* [17]: The ith occurrence of a character c in the last column corresponds to the ith occurrence of c in the first column (Fig. 7). More formally, let $C(c)$ be the total number of characters in T that are alphabetically smaller than c, and $Occ(c, i)$ the number of occurrences of character c in prefix $T^{bwt}[1, i]$. Then, character $T^{bwt}[i]$ is located in the first column at position

$$LF(i) = C\big(T^{bwt}[i]\big) + Occ\big(T^{bwt}[i], i\big). \tag{1}$$

B. Backward Search

The FM-index allows one to determine the range of lexicographically ordered suffixes of T that have pattern P as prefix without storing the corresponding suffix array. Algorithm 1, BackwardSearch, computes in iteration i the maximal range of rows in $\mathcal{M}_T$ prefixed by $P[i, m]$, or equivalently the maximal range of entries in the corresponding suffix array pointing to suffixes prefixed by $P[i, m]$.

Algorithm 1: BackwardSearch(P, m, n, C, Occ)

```
1 sp ← 1, ep ← n
2 for i ← m to 1 do
3 |   sp ← C(P[i]) + Occ(P[i], sp − 1) + 1
4 |   ep ← C(P[i]) + Occ(P[i], ep)
5 |   if sp > ep then return ∅
6 end
7 return [sp, ep]
```

The maximal range of rows in $\mathcal{M}_T$ prefixed by $P[i-1, m]$, possibly empty (line 5), can be obtained from the maximal range of rows in $\mathcal{M}_T$ prefixed by $P[i, m]$, by determining the immediate predecessors of the first characters in rows sp and ep in the original text T, which are given by $T^{bwt}[sp]$ and $T^{bwt}[ep]$, respectively, and by computing their corresponding occurrences in the first column through the *LF* mapping; see lines 3 and 4. The correctness of this relation was demonstrated by Ferragina and Manzini [33]. Fig. 8 illustrates the steps of Algorithm BackwardSearch for an example pattern.

Implementing function C as an array requires just $\sigma \log n$ bits and allows retrieval of any entry in lines 3 and 4 in constant time. Using auxiliary data structures and the *four Russians trick* [14], Ferragina and Manzini [33] showed how to find $Occ(c, i)$ in constant time too, using $\mathcal{O}(nH_k)$ bits of space. H_k denotes the kth-order entropy, a lower bound on the number of bits any kth-order compressor requires to encode a symbol. Algorithm BackwardSearch thus runs in optimal $\mathcal{O}(m)$ time.

After determining the relevant range of rows in $\mathcal{M}_T$ in $\mathcal{O}(m)$ time, the task remains to locate the corresponding starting positions in the reference genome, which would be directly given by the entries of a suffix array. Ferragina and Manzini [33] store for a suitable subset of (marked) rows of M_T their text offset. They implicitly move backwards in T by jumping between the corresponding rows in $\mathcal{M}_T$ using the *LF* mapping until a marked row is encountered. The text position returned is then equal to the text offset assigned to the final row plus the number of "jumps." A jump can be performed in constant time, and the data structure containing the text offsets of marked rows supports queries in constant time too. If every ηth row is marked, $\eta = \lceil \log^{1+\varepsilon} n \rceil$, occ occurrences of P in T can be located in $\mathcal{O}(occ \log^{1+\varepsilon} n)$ time. Bowtie [59] tries to balance the space-time tradeoff by setting $\eta = 32$, which corresponds to $\varepsilon < 0.01$. Similarly, BWA [62] samples the suffix array at intervals of size 32 and traverses the text in forward direction using the inverse function of the *LF* mapping [38].

Subsequent work on the FM index aimed at reducing its dependence on the alphabet size σ [34], [90]. The results in [33] assume that the size of the alphabet is constant, and indeed the space requirement depends exponentially on σ. Because DNA only uses four nucleotides ($\sigma = 4$), this dependence is not an issue. In the original Bowtie implementation, the BWT-based FM index for the human genome requires only about 1.3 GB of memory [59], which allows it to fit easily in the main memory (RAM) of a standard laptop or desktop computer. This also facilitates distribution of

precomputed indexes for most major model organisms; for example, the Bowtie site[5] provides prebuilt indexes for the human, mouse, dog, rat, fruit fly, and other genomes.

C. FM-Index-Based k-Mismatch Search

The indexes discussed above are designed for the exact string matching problem. Exact string matching is an idealized version of the real read mapping problem. Sequencing errors and differences between the reference genome and the sequenced organism can cause the read to deviate substantially from the the reference genome (see Section I-B2).

In the simplest case, a BWT index replaces the hash index in a scheme as discussed in Section II-A2a. When two mismatches are allowed, SOAP2 [67], for instance, splits the read into three parts such that one is guaranteed to match perfectly, and the exact match is located through the BWT of the reference genome.

Below we describe how the high-throughput read mappers Bowtie [59] and BWA [62] implement their own backtracking methods that allow for (a small number of) mismatches and that are engineered for the efficient processing of millions of short sequence reads.

1) Backtracking: The general principle of backtracking during backward search can be illustrated by considering its application to the k-mismatch problem. The pseudocode of a backtracking aware backward search that finds all k-mismatch occurrences of P in T is given in Algorithm 2, kMismatchSearch [82].

Algorithm 2: kMismatchSearch(P, k, j, sp, ep)

```
1  if sp > ep then return ∅
2  if j = 0 then return [sp, ep]
3  foreach c ∈ {A, C, G, T} do
4  |   sp' ← C(c) + Occ(c, sp − 1) + 1
5  |   ep' ← C(c) + Occ(c, ep)
6  |   if P[j] ≠ c then k' ← k − 1 else k' ← k
7  |   if k' ≥ 0 then
8  |   |   kMismatchSearch(P, k', j − 1, sp', ep')
9  |   end
10 end
```

The main difference to algorithm BackwardSearch (without backtracking) lies in line 3, where in addition to current nucleotide $P[j]$ (**else** block in line 6) all possible substitutions of $P[j]$ are considered (**then** block in line 6). The total number of substitutions is not allowed to exceed k (**if** condition in line 7). Each node of the recursion tree maintains the range of suffixes $[sp, ep]$ in the suffix array that are prefixed by the current modification of $P[j, m]$. Fig. 9 shows the recursion tree for an example search with one mismatch, i.e., $k = 1$.

[5]http://bowtie-bio.sourceforge.net/index.shtml

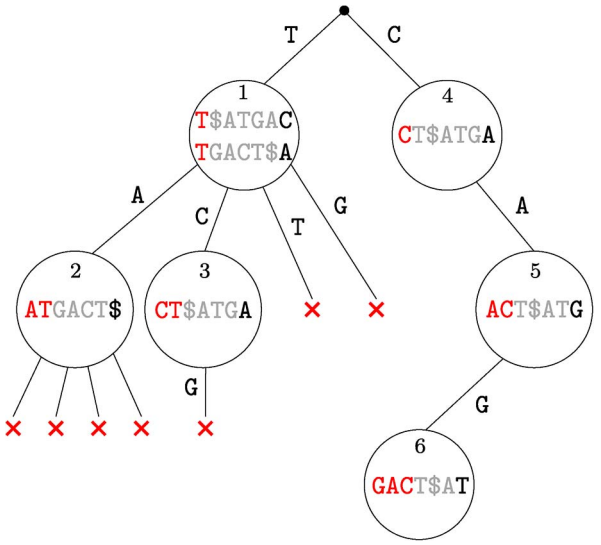

Fig. 9. *Recursion tree as traversed by Algorithm kMismatchSearch for $P = \mathtt{GAT}$, $T = \mathtt{ATGACT}$, and $k = 1$. Nodes contain strings falling in current range $[sp, ep]$ and are expanded in depth-first manner as indicated by their numbering. Red crosses denote pruning in line 1 of the algorithm. Notice that for $c \neq \mathtt{G}$ the condition in line 1 evaluates to false in node 3. When reaching node 6, an occurrence with 1 mismatch has been found.*

BWA [62] generalizes algorithm kMismatchSearch in a straightforward way to find *k-errors* occurrences, allowing for both mismatches and insertions and deletions. Note that the practicability of Algorithm 2 strongly depends on the alphabet size since all possible substitutions are considered in line 3. In other words, current read mapping methods that apply the FM index through a backtracking procedure to the inexact matching case exploit a small alphabet size of $\sigma = 4$.

2) Engineering Backtracking: The worst-case running time of kMismatchSearch is $\mathcal{O}(|\Sigma|^k m^{k+1})$ [83]. To further improve their practical performance, Bowtie [59] and BWA [62] invest some additional memory to not only build the FM index for the genome T, called the *forward* index, but also for the reverse (not complemented) genomic sequence $T^r = t_n t_{n-1} \cdots t_1$, called the *reverse* or *mirror* index. However, the two methods exploit the reverse FM index in different ways, as we will illustrate in the following sections. In Section III-C2d, we briefly summarize algorithm design decisions that are influenced by specific properties of the data such as base quality values and systematic errors.

a) Pruning by bounding: BWA uses the reverse index to precompute for each prefix $P[1, i]$ of a given read P the minimum number of parts $D[i]$ into which $P[1, i]$ has to be broken into such that each part occurs at least once in genome T (exactly) [82]. Array D can be computed in time linear in the read length through a backward search of the

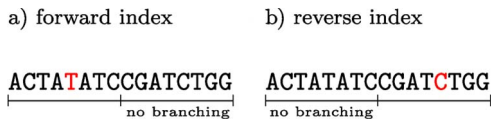

Fig. 10. *To minimize backtracking, the search direction in Bowtie depends on the location of the mismatch (red).*

reverse read in the reverse FM index, increasing the current entry of D by one every time range $[sp, ep]$ becomes empty. $D[i]$ provides a (not necessarily tight) lower bound on the number of errors an approximate occurrence of $P[1, i]$ in T exhibits and can thus be used to prune the space of potential matches explored by backward backtracking (see Section III-C1) as follows. Let e_v be the total number of errors introduced in line 3 of Algorithm 2 up to current node v of the recursion tree reached after backward searching suffix $P[j, m]$. Then, if $e_v + D[j-1] > k$ proceeding the search from node v cannot yield a match with at most k errors and therefore the current branch can be pruned.

b) Pruning by case sensitivity: The pruning strategy of Bowtie can be seen as an extension of the pigeonhole principle underlying the seed-and-extend paradigm described in Section II. The *case-sensitive backtracking* scheme considers all possibilities to distribute k mismatches to two segments of the read and disallows backtracking in a segment that is assumed to be error-free in the current mismatch scenario. To prune the search space, backtracking on higher levels (i.e., lower depth) of the search tree is avoided by searching read P from right or left using forward or mirror index, depending on where the error-free segment of the read is assumed to lie. This idea can be illustrated for the case where only one mismatch is allowed in the alignment ($k = 1$) and the read is split into a left and a right half (Fig. 10). The mismatch can lie either in the left half or in the right half of the read. In the first case [Fig. 10(a)], the right half must be error-free, and therefore in line 3 of Algorithm 2 no substitution is necessary, that is, c must be equal to $P[j]$ if j lies in the right half of the read. Searching P from right to left using the forward index restricts branching to lower levels (i.e., higher depth) of the search tree. The second case [Fig. 10(b)] is a mirror of the first case and thus P is searched from left to right using the mirror index, disallowing substitutions in the left half of the read and thus again avoiding branching on higher levels of the search tree.

For two or more mismatches ($k \geq 2$), not every assignment of mismatches to read segments implies an error-free segment. In fact, for every $k \geq 2$ there is a worst-case scenario where both read segments have at least $\lfloor k/2 \rfloor$ erros. Bowtie switches in different *phases* between forward and mirror indexes, depending on where the segment assigned a smaller number of errors by the current scenario lies. Again, such a segment is preferably treated on higher levels of the search tree.

c) Pruning versus filtering: In [82] the pruning heuristics introduced in Section III-C2a and C2b were experimentally evaluated on ChIP-seq data taken from [110] comprising 10 000 36-bp-long reads. For a varying number of mismatches k, these heuristics were compared against the *suffix filter* [48], a state-of-the-art *filtering* algorithm (see Section II) developed for the approximate string matching problem. The case-sensitive backtracking strategy implemented by Bowtie slightly outperformed the suffix filter in terms of search time per read, when either the best, or all occurrences were located. A novel pruning heuristic proposed in [82] that combines suffix filter and case-sensitive backtracking was superior in speed, at the cost of a significantly higher space consumption.

d) Application engineering: Compared to the straightforward recursive scheme applied by Algorithm 2, Bowtie and BWA exploit the quality of a partial alignment, i.e., its score, to guide the (implicit) traversal of the suffix tree. BWA uses a heap structure to implement a *best-first* strategy, Bowtie traverses the tree in a slightly modified *depth-first* manner, selecting greedily substitution positions with minimal quality value.

Both methods employ a *seeding* strategy (see Section II) that assumes a small number of sequencing errors in a high-quality part of the read, which usually comprises a few tens of bases at the 5′ end of the read. A smaller number of errors reduces the need to backtrack while backward searching the seed and allows to find promising candidate regions on the genome efficiently. For example, the effectiveness of Bowtie's pruning strategy (see Section III-C2b) depends on the existence of a read segment with a small number of errors (ideally error-free). Therefore, the division into halves is with respect to the seed instead of the full read.

Several restrictions are imposed to further balance the efficiency-sensitivity tradeoff. For example, the total number of errors and backtracks allowed is limited by BWA and Bowtie, respectively. A relatively low error rate exhibited by the read sequences and a typically low divergence between the sequenced individual (from which reads P are derived) and the reference genome T render this tradeoff suitable for most applications.

D. FM-Index-Based Similarity Search

Lam *et al.* [56] describe an extension of BackwardSearch to the search for most similar occurrences of P in T that may exhibit mismatches, insertions and deletions. Gaps are penalized by an affine function in their length. Insertions and deletions occur more often in longer reads produced by current sequencing technology. In particular, indels are the predominant type of sequencing error in data generated by single-molecule sequencing technologies [53].

Their tool BWT-SW, which is designed for the alignment of relatively long queries (e.g., 3000 bp) against the human genome, resorts to local alignments computed by

dynamic programming. The main difference to the Smith–Waterman algorithm lies in the (implicit) suffix trie representation of T against which pattern P is aligned to. Roughly speaking, when the alignment of P and a path in the suffix tree reaches a node v, P has implicitly been aligned to all occurrences of the string represented by v in T, due to the common prefix structure of the trie. Representing T by a BWT, BWT-SW generates all words in T by traversing the suffix trie in a preorder fashion (backtracking), determining edges on the fly using backward search.

The search can be restricted to *meaningful* alignments [56], which allows to prune the search space without sacrificing optimality. Overall, in their experiments, Lam *et al.* observed a 1000-fold speedup compared to the Smith–Waterman algorithm. For longer patterns however, BLAST ran several times faster than BWT-SW and missed only few significant alignments. In general, the time required to compute an alignment increases to $\mathcal{O}(m^2)$, compared to $\mathcal{O}(m)$ if only mismatches are considered, and renders an exhaustive backtracking-based approach infeasible for the read mapping problem.

To find approximate occurrences of a read in a more general error model including (affine-weight) gaps efficiently, Bowtie 2 [57] and BWA-SW [63] resort to the seed and extend paradigm that was already applied successfully by earlier methods discussed in Section II. The crucial difference lies in the seeding step, where candidate mapping locations on the genome are determined efficiently using the FM index. While BWA-SW permits gaps in the seed alignments, Bowtie 2 tries to "guess" a short substring of the read that aligns with few mismatches only. This anchoring idea is taken one step further by Chaisson and Tesler [19], who developed BLASR to address the computational challenge posed by aligning multikilobase long reads generated by single-molecule sequencing technologies with a high error rate. We summarize the main algorithmic concepts of BLASR and BWA-MEM, which complements the BWA software package [62], [63] for the scenario of long-read alignments, in Section III-F.

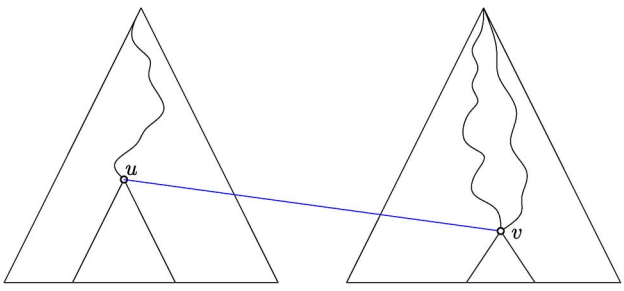

Fig. 11. *Alignment of a prefix trie representation T of the genome (left) and a DAWG G capturing the read sequence (right) by BWA-SW [64]. Starting at the roots of the graphs, the DP recursion relates the optimal alignment of substrings represented by a node u in T and a node v in G to the optimal alignment of substrings expressed by their parent nodes. Node u in T and v in G represent all occurrences of the string spelled by the (unique) u-to-root path in the genome and all occurrences of the strings spelled by the v-to-root paths in the read, respectively. The v-to-root paths in G by construction spell substrings of the read such that one is a prefix of the other.*

AATCCTAGGACTACGACCAGTAGCTAGCG

Fig. 12. *Seed Strings in Bowtie 2: In this example, 13 nt substrings spanned by the lines are extracted every 8 nt.*

1) Gapped Seeds: Li and Durbin [63] combine and extend the dynamic programming scheme on suffix tries from [56] and the heuristic seed and extend paradigm of BLAST [7]. BWA-SW determines seeds by a dynamic program between a prefix trie and a prefix directed acyclic word graph (DAWG) [12], allowing mismatches and indels. A prefix DAWG is obtained from a prefix trie by collapsing nodes that correspond to the same suffix array interval and is implemented, as the prefix trie, by an FM index. In that way, the evaluation of a single pair of nodes essentially compares a set of substrings in T with a set of substrings in P, exploiting repetitiveness both in T and in P. See Fig. 11 for an illustration. As soon as these sets become too small, the overhead involved in traversing prefix trie and prefix DAWG exceeds the benefits and these seeds are extended by the Smith–Waterman algorithm. Two heuristics are applied to accelerate the computation. The DP is pruned at low scoring nodes, and only seeds that are likely to lead to distinct alignments are retained.

2) Ungapped Seeds: Bowtie 2 [57] relies on seed strings, i.e., possibly overlapping (consecutive) substrings of P, obtained from evenly spaced intervals along the read. See Fig. 12 for an example. The seed strings are chosen to be short enough (e.g., 20 nt) that their alignment can ignore indels and has to account only for a small number of mismatches (≤ 1). Such an alignment can be computed efficiently by an FM index supported backward search with very limited backtracking (see Algorithm 2). In this step, Langmead and Salzberg apply a scheme similar to the one developed in Bowtie [59], with a bidirectional BWT [57] supporting an efficient switch between alignment directions. Seed alignments are selected in an order that favors those with a smaller number of alternative occurrences in T and are subsequently extended by a hardware-accelerated dynamic program. SIMD instructions, supported by most of the modern CPUs, are used to parallelize the Smith–Waterman algorithm. It organizes the data for the vector instructions similarly to [30], with slight adaptions to the specifics of the read alignment problem.

E. FM-Index-Based k-Error Search

As mentioned in Section II-A2a, the seeding strategies of Masai [105] and the GEM mapper [84] aim at full

sensitivity by employing the pigeonhole principle, according to which there always exists a seed with at most $\lfloor k/s \rfloor$ errors in a partitioning of a k-error occurrence into s seeds.

The GEM mapper determines the partitioning adaptively based on the number of matches of each seed in the genome. During backward search in the FM-index a new seed is started as soon as the number of matches ($ep - sp + 1$, see Section III-B) of the current seed falls below a certain threshold, keeping the number of candidate regions to be verified small. If the resulting number of seeds is large enough ($s \geq k + 1$), full sensitivity can be guaranteed by searching for exact seed matches, otherwise approximate matches of seeds have to be taken into account.

Masai searches for a seed with at most $\lfloor k/s \rfloor$ errors through backtracking [111] in a (conceptual) suffix tree of the reference genome, implemented as a suffix array or FM-index. A second index organizes the nonoverlapping seeds of the reads in a radix tree and allows to search all seeds in the suffix tree simultaneously. Alternatively, Masai permits to partition the read into $s = k + 1$ seeds per read and searches for exact matches using a simpler and more efficient algorithm.

Notice that both the filtration and in particular the extension based on Myers bit-vector algorithm [88] (see Section II-C2) as employed by Masai and GEM rely on the edit distance measure and do not take into account quality scores.

F. FM-Index-Based Long-Read Alignment

Chaisson and Tesler [19] tailor the idea of anchoring the alignment at multiple short (exact) seed matches to the problem of mapping longer and error-prone reads produced by single-molecule sequencing methods. Aligning reads several kilobases long containing a higher number of errors, most of which are insertions and deletions, poses a computational challenge that shifts the NGS alignment problem slightly towards the problem of aligning whole genomes (WGA) [23], [102]. Their tool BLASR therefore combines a successive refinement strategy typically employed in WGA with space- and time-efficient data structures that can yield a boost in alignment speed. It applies a distance measure that does not simply count the number of edit operations, but assigns alphabet-dependent weights.

BLASR first finds candidate intervals on the reference genome that are subsequently evaluated by a pairwise alignment routine. Candidate intervals resemble dense clusters of short exact matches between read and genome that are consistent in terms of coordinate range, order, and orientation. Exact matches are obtained by scanning the read from left to right and searching the longest common prefix of the current read suffix and the genome using either a suffix array or BWT-FM index. The resulting anchors are clustered by chaining [2] a maximal set of non-

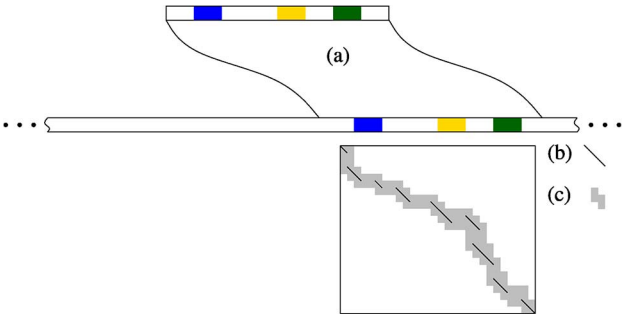

Fig. 13. *Three main steps performed by BLASR [19]: (a) Candidate genomic regions are identified by clusters of consistent exact matchtes. (b) A sparse dynamic program prealigns the read to the highest scoring candidate regions. (c) Guided by the anchors found through the sparse alignment, a final base-resolution alignment is computed.*

overlapping anchors in increasing order (by coordinate in genome and read) without exceeding the read length.

The anchor chains define the intervals to which the read is aligned, extending the boundaries slightly to take into account insertion and deletion errors that blur the start or end position of the read [Fig. 13(a)]. The pairwise alignment of the read and the intervals is further subdivided into a more coarse alignment using sparse dynamic programming (SDP) [28] [Fig. 13(b)] and a more comprehensive banded alignment guided by the coarse alignment [Fig. 13(c)]. SDP again determines a consistent set of small exact matches (anchors) between the read and the genomic interval. The final dynamic programming considers only entries in the DP matrix that lie in a band around the anchors returned by the previous SDP step. The scoring of the alignment is based on the quality values and the alternative base calls provided by the PacBioRS platform.

The feasibility of this approach clearly depends on: 1) a sufficiently high number of anchors marking the true mapping location, and 2) an overall low number of anchors lying outside this true genomic interval. Chaisson and Tesler propose a method to computed the probability of event 1) for a given read length and sequencing accuracy on the one hand, and study the repetitive structure of the human genome to assess 2) on the other. They conclude that for the human genome, both conditions are satisfied, rendering their successive refinement strategy feasible in this context.

BWA-MEM, available as part of the BWA software package [62], [63], also relies on maximal exact matches to seed alignments of longer and error-prone reads generated by, e.g., the PacBioRS platform. Compared to the longest common prefix that BLASR computes for every read suffix, BWA-MEM restricts the search space to supermaximal exact matches (SMEMs) between reference and read, i.e., exact matches that cannot be extended at either

end and that are not contained in any other exact match on the read. The unique variants of SMEMs were previously used by Delcher *et al.* [24] as maximal unique matches (MUMs) to anchor the alignment of two whole genome sequences.

BWA-MEM's seed extension uses a banded Smith–Waterman algorithm with heuristic modifications that increase its robustness to sequencing errors: A suboptimal end-to-end alignment can be chosen heuristically over a local alignment that achievs a higher score, and extensions through low-scoring regions are avoided without penalizing long gaps in either of the sequences.

IV. AGONY OF CHOICE

Despite several algorithmic advances (e.g., spaced seeds) that benefitted hashing-based methods, this class of methods were largely replaced in practice by tools relying on the FM index, after Bowtie and BWA first introduced this space- and time-efficient data structure in the context of short read mapping. In the experiments performed in Langmead *et al.* [59], Bowtie mapped 35-bp reads 38–350 times faster than state-of-the-art hashing-based methods, with almost no loss in sensitivity, making such methods the choice of short read aligner for large-scale mammalian resequencing studies.

The low error rate of high-throughput sequencers together with a typically low divergence between reference and sequenced organism allowed aligners like Bowtie and BWA to traverse the space of mutations for very short reads efficiently. With read lengths increasing to 100–150 bp, however, the alignment throughput of purely index-based methods decreases. Bowtie 2 and BWA-SW therefore explore the larger search space of longer, gapped alignments by resorting to the seed-and-extend paradigm.

Much longer and more error-prone reads generated by recent single-molecule sequencing technologies, with lengths exceeding 10 000 bp and error rates of 15% or higher, pose new challenges to read mapping tools. With insertions being the predominant type of sequencing error of this technology [53], error models that neglect or restrict gaps are inadequate. Algorithmic recipes successful in this context combine and generalize concepts developed in the short read mapping and the whole-genome comparison literature. To avoid incurring quadratic cost for aligning a long read, algorithms can anchor the alignment at multiple seeds that ideally correspond to maximal stretches (e.g., longest common prefix [19], MUMs [24], or SMEMs as used in BWA-MEM) of error-free nucleotides. These seeds can be matched efficiently using a structure like the FM-index and guide a coarse alignment method [19] developed for the alignment of whole genomes.

Although these observations suggest a certain class of alignment method to be applied in a particular scenario, they typically leave the user with a large number of mapping tools to choose from. As previous benchmarks showed (see [40] and references therein), alignment sensitivity and throughput also varies with quantitative data parameters such as read length, genome size, and genome repetitiveness, and there is no single best aligner. Moreover, the choice is not only among available tools, but also among the many parameters that can be provided to adjust an algorithm to the characteristics of the data, available hardware, and the requirements imposed by the downstream analysis. For example, BWA-MEM offers options to adjust the scoring scheme, seed length, and filtering strategy when run on reads generated by particular sequencing technology. The aim of this review is not to suggest the use of specific tools, but rather to enable and guide the reader to select or design mapping software and parameters that best suit a particular need.

V. PARALLELIZATION

The enormous data sets generated by modern sequencing machines require greater speed than any of the methods described above can deliver on a single processor. To address this challenge, many methods implement at least one form of parallel computing. The widely used aligners Bowtie1/2 [57] and BWA [62] include built-in methods to distributing work (e.g., reads) across multiple threads to make use of multicore CPUs or multi-CPU nodes. Taking parallelization one step further, CloudBurst [100] and Crossbow [58] are two prominent examples of systems that utilize *cloud computing* for parallel read alignment implementations. Using Hadoop,[6] an open-source implementation of Google's Map-Reduce programming model [23], these two systems parallelize the execution of RMAP and Bowtie, respectively, using large clusters of commodity hardware.

A cost-effective alternative to cope with the plateau in CPU clock speeds is to leverage the massive parallelism provided by a GPU, whose high-level architecture is shown in Fig. 14. Designed primarily to perform computer graphics calculations, GPUs are optimized for throughput rather than latency. On a GPU, hundreds of high-performance streaming processors execute small tasks in parallel. This processing power can also be harnessed by data-parallel algorithms developed for diverse general-purpose applications (known as General Purpose computing on GPUs, or GPGPU), including the alignment of short NGS reads to a reference genome.

GPU-based aligners rely on the same general principles discussed in Sections II and III, such as the seed-and-exend paradigm, searching an FM-index for approximate occurrences through backtracking, or the verification of candidate genomic regions through a Smith–Waterman alignment. However, exploiting the computing power of GPUs in the highly parallel alignment of short reads involves more than simply assigning to each GPU thread

[6]http://hadoop.apache.org

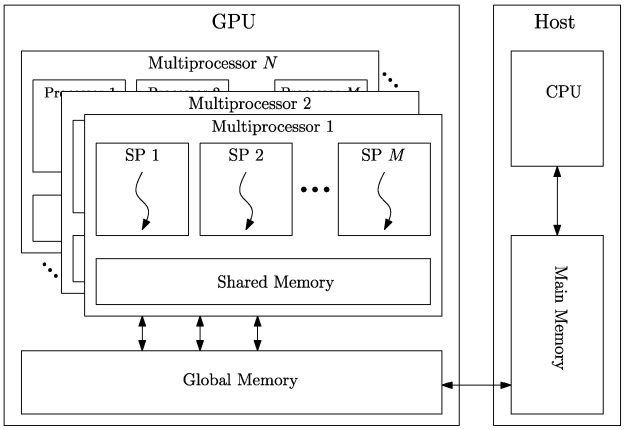

Fig. 14. *Architecture of a GPU. GPUs comprise a number N of multiprocessors that can simultaneously access the global memory. Multiprocessors consist of several stream processors (SP), or cores, that can all access the shared memory located on the same multiprocessor. The cores execute the sequential threads (curved arrows) in SIMT mode (see text).*

a single read and launching thousands of threads. The massive parallelism of GPUs comes at the cost of a more restrictive programming model compared to CPUs. From a software development point of view, the following properties of the GPU architecture determine the design of data structures and algorithms.

1) Similar to classical SIMD pocessors, GPU cores execute sequential threads on different data in a *single instruction, multiple thread* (SIMT) fashion. The hardware groups several threads together that execute the same instruction at the same time, on different operands.
2) Repeated accesses to the global memory and space-hungry data structures can cause memory contention in the highly parallel environment of a GPU.
3) Global memory access is considerably slower than performing arithmetic operations.
4) Many consecutive locations in global memory can be accessed in parallel.

Furthermore, the engineering of GPU-based software requires one to take into account features of the underlying hardware such as shared memory and cache (hierarchy) configurations. Although these aspects can have a large impact on the overall performance of the software, we focus on the algorithm design decisions made by current GPU aligners to adhere to the data-parallelism properties listed in 1–4 above. The hashing-based methods in the following section divide the work between CPU and GPU in a straightforward way, while BWT-based methods in Section V-B make more explicit use of the GPU properties 1–4.

A. Hashing-Based GPU Aligners

Blom *et al.* [11] developed SARUMAN, the first short read aligner that makes use of GPU computing power. In their algorithm, the host CPU precomputes a hash index from the reference genome and performs filtration based on the pigeonhole principle; see Section II-A2a. The resulting candidate mapping locations are verified in parallel through GPU threads that compute a global alignment by a modified Needleman–Wunsch algorithm.

Similarly, GPU-RMAP [5] builds on RMAP (see Section II-A2a) and indexes the reads on the host. In contrast to SARUMAN, however, the genome is divided into independent segments and GPU threads scan these segments in parallel to find and score (number of mismatches) mapping locations of all reads. In a second stage, reads are distributed to GPU threads that select the best scoring location. Compared to the sequential implementation of RMAP [108], GPU-RMAP replaces the hash table containing the reads by a binary search tree, placing frequently accessed levels of the tree into faster cache memory.

B. BWT-Based GPU Aligners

BarraCUDA [51], SOAP3 [73], SOAP3-dp [80], CUSHAW [77], and CUSHAW2-GPU [75] are all based on the BWT and variants of the FM-index (see Section III), and each employs different strategies to address GPU properties 1–4:

1) Avoiding Divergence of Execution: A consequence of property 1 is that divergent execution paths (branching) of threads in the same group drastically harms performance. The exploration of potential substitutions (see backtracking in Section III-C1) will lead to many branches for some reads that will cause other cores with few branches to be idle and wait for the other threads to reach the same point of execution. To avoid this loss in parallel efficiency, SOAP3 and SOAP3-dp align reads in groups of similar branching complexity, which is estimated at runtime from the number of suffix array ranges. The most complicated reads are handled by the CPU. For the same reason, BarraCUDA divides reads into segments and explores potential (inexact) matchings by consecutive depth-first searches, each restricted to a segment.

2) Reducing Memory Consumption: One obvious way to cope with the limited device memory is to organize reads or genomic candidate regions in batches that are scheduled one by one (CUSHAW, CUSHAW2-GPU).

To reduce the memory consumption per thread (see property 2), BarraCUDA and CUSHAW traverse the space of possible base substitutions and indels (BarraCUDA only) in a depth-first search manner. During DFS, BarraCUDA trades efficiency for reduced memory consumption by storing only the currently best branch, evaluating certain nodes multiple times. CUSHAW further prunes the search space by imposing additional constraints on the overall quality score.

Compared to the jumping strategy proposed by Ferragina and Manzini [33] to locate a read occurrence on the genome, CUSHAW and CUSHAW2-GPU avoid

auxiliary data structures by resorting to a reduced suffix array that stores only every ηth entry. In contrast to [33] however, a search cannot be guaranteed to be successful after η jumps. Similarly, CUSHAW stores *Occ* (see Section III-A) only at every 128th position and interpolates the remaining entries through T^{bwt} as needed.

3) Reducing and Coalescing Memory Access: SOAP3 and SOAP3-dp reduce random memory access (properties 2 and 3) required by the 2way-BWT index [55] that facilitates the switch in search direction during the approximate matching of a read. In particular, an auxiliary array supporting $Occ(c, i)$ queries (see Section III-B) is built following a simple one-level sampling strategy. The sampling ratio is chosen such that: a) a single access to T^{bwt} exploits the full memory bus width; and b) the organization of the auxiliary array into groups of *Occ* values for the same position allows to determine $Occ(c, i)$ for all four nucleotides c through a single memory access (see property 4). The latter operation is of key importance for the efficient search in the 2way-BWT index.

To accelerate memory access (property 3 and 4), SOAP3 and SOAP3-dp *coalesce* simultaneous access of different threads from the same SIMT group to global memory. The idea is to arrange reads in the memory in a way such that memory requests of threads in the same SIMT group can be satisfied in a single memory transaction. For that, reads are partitioned into groups of the same size as there are threads in a SIMT group and the jth words of the reads in the same group occupy a consecutive region in memory (see Fig. 15). The access to the BWT index in global memory is more random and thus can be coalesced only to a small degree.

The strategies discussed so far mostly involved adapted data structures and memory orgranization. From an algorithmic point of view, few memory accesses are desired (properties 2 and 3) when verifying a candidate region using a Smith–Waterman algorithm. SOAP3-dp applies a GPU-tailored variant of the Smith–Waterman algorithm [74] that reduces the number of memory operations per iteration. Similarly, CUSHAW2-GPU computes alignment scores using a slightly modified variant of a GPU-accelerated SW algorithm [76] and reconstructs the best scoring alignments by backtracking in 4×4 tiles. Since 2 bits suffice to encode the three possible edit operations in each cell, an alignment restricted to a tile can be represented by a single 32-bit integer and thus proceeding in units of a tile is supported by single read/write operations to the matrix storing the corresponding edit operations.

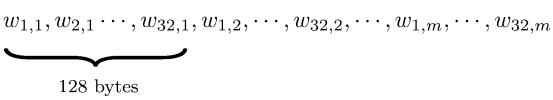

Fig. 15. *Arrangement of reads in memory by SOAP3 and SOAP3-dp. For $1 \leq j \leq m$, the jth 4-B word w_{ij} of all reads i in a group of 32 reads occupy a contiguous 128-B region, which can be retrieved by a single memory access operation.*

C. GPU Versus CPU Aligners

A general statement about the relative performance and practicability of GPU-based and CPU-based aligners is difficult to make since it would be based on a comparison of algorithms that run on hardware implying a different cost and availability. Nevertheless, the gain in alignment throughput of GPU-based tools over methods that run on multiple cores on the CPU has been rather modest. While BarraCUDA and CUSHAW achieved a throughput that was comparable to that of BWA run on 6 cores [51] and Bowtie using 1–4 threads [77], respectively, they were outperformed [80] by CPU-based aligners like Bowtie 2 [57] and GEM [84]. The successor CUSHAW2-GPU achieved a 1.5-fold speedup over Bowtie 2 when employing one GPU in addition to the 12 CPU threads Bowtie 2 used. SOAP3-dp seems to utilize the computing power of GPUs most efficiently. In their experiemts [80], SOAP3-dp ran 3.5 times faster than GEM and 7 times faster than Bowtie 2 when using one GPU device in addition to 4 CPU threads.

Wilton *et al.* [117] identify two main obstacles that hinder GPU-based alignment software to profit more directly from the massive parallelism provided by a GPU. First, the Smith–Waterman dynamic program formulates dependencies that favor the independent computation of entire alignments in parallel compared to a data-parallel implementation of the algorithm suitable for GPUs. Second, few seeds extracted from reads match a very large number of subsequences in the reference, implying an enormous computational burden for the alignment software. In [117] the authors propose "GPU-friendly" heuristics based on list-manipulation operations for pruning this search space. By assigning those components of the sequence alignment pipeline to the GPU that are amenable to a data-parallel implementation, Arioc [118] achieves a 10-fold speedup over state-of-the-art CPU-based aligners Bowtie 2 and BWA-MEM.

We believe that GPU-based aligners will play an important role in the analysis of datasets generated by large-scale sequencing studies involving thousands of individuals. GPU-based aligners scale well with an increasing number of GPU devices used [51], [117] and thus provide a means to support GPU-tailored algorithmic advances with additional hardware [117].

VI. MAKING AN EDUCATED GUESS

The distance or similarity threshold k is generally chosen such that the true origin of a read is likely to be among the approximate occurrences implied by k. Read mappers like RazerS (3), Zoom, Hobbes, GEM, Masai, and BWT-SW simply enumerate and report *all* occurrences, leaving it to

the downstream analysis to pick the right location. They sometimes allow for an adjustable sensitivity or limit the number of reported occurrences of highly repetitive sequences.

A. Picking the Best Mappings

Alternatively, several approaches have been proposed that make an attempt to further narrow down the set of potential mapping locations, or even guess the true read origin as the best alignment. In the simplest case, the alignments are ranked by the distance or similarity score. For instance, SOAP (2) by default returns the alignment imposing the minimal number of mismatches or the smallest gap. Similarly, BFAST, BWA, and Bowtie 2 (default mode) prioritize alignments achieving the highest score when guessing the true read origin, while Shrimp outputs the best n scoring alignments. BLASR only considers up to a certain number of the least repetitive candidate genomic regions.

Methods that are based on a Hamming distance or edit distance rather than a more general scoring scheme often take into account base quality scores to guide the subsequent prediction of the read origin. MAQ, for example, picks the location where the sum of the qualities at mismatched bases is minimized. Accordingly, Bowtie greedily seeks alignments with low qualities at mismatched positions. BWA-SW, on the other hand, aims at selecting distinct ones among the highest scoring alignments.

B. Estimating the Qualtiy of a Mapping

Some real aligners provide, with each alignment, an estimate of the probability that the alignment is the true genomic origin of the read. This is not a measurement of the alignment quality itself: It does not depend on the number of mismatches, consistently mapped read pairs, or the quality of the sequenced DNA. Rather, it captures the uncertainty that arises when a read's true source is a repetitive region of the genome. If a read can be mapped equally well to two different genomic locations, then in statistical terms we can only have a 50% probability of assigning it to the correct location. Downstream analysis protocols, especially variant calling in human DNA sequencing, typically rely on these mapping qualities to decide which variants can be called with confidence. Li *et al.* [64] introduced the mapping quality score Q (or MAPQ) and scaled it [29] as

$$Q = -10 \log \Pr[\text{mapping is incorrect}]$$

i.e., a quality value of 20 corresponds to a probability of 0.01 that the mapping is incorrect. Higher quality values indicate higher confidence in the mapping.

Assuming that genomic positions are chosen by the sequencer with equal probability (uniform prior distribution), the posterior probability of a read sequence R originating from a position m in the reference sequence G is

$$\Pr[m|R, G] = \frac{\Pr[R|G, m]}{\sum_{i=1}^{|G|} \Pr[R|G, i]}. \tag{2}$$

The numerator gives the probability of obtaining a particular sequence R when sequencing begins at position m in the reference. It depends on the divergence between reference and sequenced individual and the error rate of the sequencing process. Typically the former is ignored and reads are treated as being obtained from the reference. The latter is ideally captured by the scoring scheme that the alignment algorithm optimizes for. While the likelihood of a mismatch to be caused by a sequencing error can be estimated from the base quality scores (see Section I-A1), the scoring of indels usually is more empirical. MAQ [64], for instance, minimizes the sum of quality scores of mismatched bases. Bowtie 2 [57] additionally applies an affine gap penalty. The banded dynamic program employed by BLASR [19] explicitly takes into account quality values provided by PacBioRS for substition, insertion, and deletion events. SHRiMP [97], on the other hand, estimates the rate of mutations (substitutions, indels), and sequencing errors via bootrapping. Note, however, that these estimates play a role only in the assessement of mapping quality; the alignment algorithm optimizes a different scoring scheme.

Algorithms computing one (arbitrary) best alignment in terms of alignment score only capture the similarity in the sense of the numerator in (2). To provide a mapping quality for an alignment, its uniqueness is measured by the denominator of (2). It captures the probability that read R was obtained from anywhere in the genome. Since summing $\Pr[R|G, i]$ over all genomic positions i is computationally infeasible, existing methods only consider the sums from genomic regions exhibiting a high similarity to R. For instance, MAQ estimates the uniqueness character of an alignment from the best alignment and all second best alignments. Bowtie 2, BLASR, BWA [62], and SOAP3-dp [81] apply a similar uniqueness criteria in assigning mapping qualities as MAQ. In contrast, SHRiMP counts the number of substrings of a random genome of the same length to which a read can align with the same number of mismatches to guess the uniqueness of its alignment on the genome.

Note that computing a mapping quality imposes a significant cost on alignment. For example, if a user has specified that the program should only return the single best alignment (which is a very common usage), it can halt immediately when it finds a perfect match to a query. However, if the program must return a mapping quality, it needs to search—in every case—for at least one more alignment in order to estimate the likelihood that the read

should map to a different location. Therefore, mappers that do not provide alignment qualities or that use a random genome as proxy, like SHRiMP, have an inherent speed advantage, but they do not provide a crucial output that many downstream analysis systems require.

VII. BEYOND STRING MATCHING

A. Paired-End Reads

One important aspect in which the task of mapping short reads differs from the (approximate) string matching problem as modeled in Section I-B is the availability of read pair information. As mentioned in Section I-A1, current instruments typically sequence both ends of the original DNA fragment, providing valuable information concerning the relative orientation and distance of the mates.

Some read mappers align both mates independently and then employ the pairing constraints to guide the selection of the true read origin among a set of candidate alignments. More specifically, Bowtie 2, BWA, SOAP2, MAQ, GEM, ZOOM, and CUSHAW2 constrain a pair of mates to align in a consistent orientation and separated by a distance that is concordant with an empirically estimated fragment length distribution. Pairs of alignments that violate these constraints are either marked as discordant in the output, or removed entirely. The orientation and distance constraints often help to resolve a repetitive read if its mate can be aligned unambiguously. BWA-MEM goes one step further and selects one best pair based on a score that considers orientation and distance of the mates, as well as the individual alignment scores.

Besides improving the accuracy of the mapping, read pair information can also be used to reduce the computational cost of alignment. In one strategy, candidate alignment locations are verified only if they satisfy the read pairing contraints (e.g., RazerS, Hobbes). In another strategy, a mapped read defines an anchor that restricts the search for aligning its mate to a limited genomic region (e.g., Bowtie 2, CUSHAW). The latter approach also allows the aligner to spend more resources aligning reads by applying the computationally expensive Smith–Waterman alignment algorithm to the genomic region defined by a successfully mapped read and the likely distance to its mate (e.g., BWA-SW, BWA-MEM, GEM, SOAP3-dp, CUSHAW2, CUSHAW2-GPU).

B. Horizontal and Vertical Integration of Alignments

1) From Pairwise to Multiple Alignments: Another simplification that is implied by the (approximate) string matching abstraction is the *independent* alignment of individual reads or read pairs. A single base in the genome is usually contained in more than one read. The read alignment problem can therefore be seen as a *multiple sequence alignment* problem. Since a global multiple alignment of all reads is computational infeasible, and since most of the reads do not overlap, the simplification to pairwise alignments employed by all current read alignment tools is well justified.

Nevertheless, the context of multiple alignment can be used in *local realignments* to improve the accuracy of the mapping. For instance, in contrast to sequencing errors, a true variant in the sequenced individual compared to the reference is expected to cause read alignments spanning this variant to *consistently* indicate the variant. While an independent *pairwise* aligment of reads cannot exploit such a signal, a local realignment (see GATK [25]) performing a *multiple alignment* of sequences in the vicinity of the variant in question (particularly indels) can partially distinguish between true variants and sequencing errors.

Similarly, Scalpel [89] combines the information provided by multiple reads through a microassembly of reads initially mapped to regions of interest (e.g., exons). Based on a gapped alignment of the assembled sequences, insertions and deletions can be called with higher accuracy.

2) Vertical Integration: TotalReCaller [85] employs the reference genome sequence during base calling (see Section I-A1) by simultaneously aligning the partially generated read sequence that grows base-by-base. This integration of base-calling and alignment decreases the error rate of the former which in turn allows to align more reads back to the reference. The resulting bias towards the reference genome, however, might lower the sensitivity in detecting true single nucleotide polymorphisms.

VIII. CONCLUSION

At least 70 read mapping tools (see Fig. 16 for examples) have been published since the advent of NGS technology. Although many of them make valuable theoretical and methodological contributions, only very few software tools are routinely used in the analysis of the enormous amounts of NGS data that are being generated at an ever-increasing speed throughout the world. The reasons are manifold, but the overarching challenge is to develop sophisticated computational algorithms that also address the practical constraints imposed by DNA sequencing technology and the vagaries of biological experimentation.

The first and most obvious criterion for the practical utility of a read mapper is its *efficiency*, both in time and space. For these very large data sets, even constant factors matter, and an algorithm that runs twice as fast can save literally years of CPU time. Space matters as well, in part because the most common "text," the human genome, is 3 billion bases long. An index that fails to represent the text efficiently may require tens of gigabytes of real memory, an amount that exceeds the standard memory available on

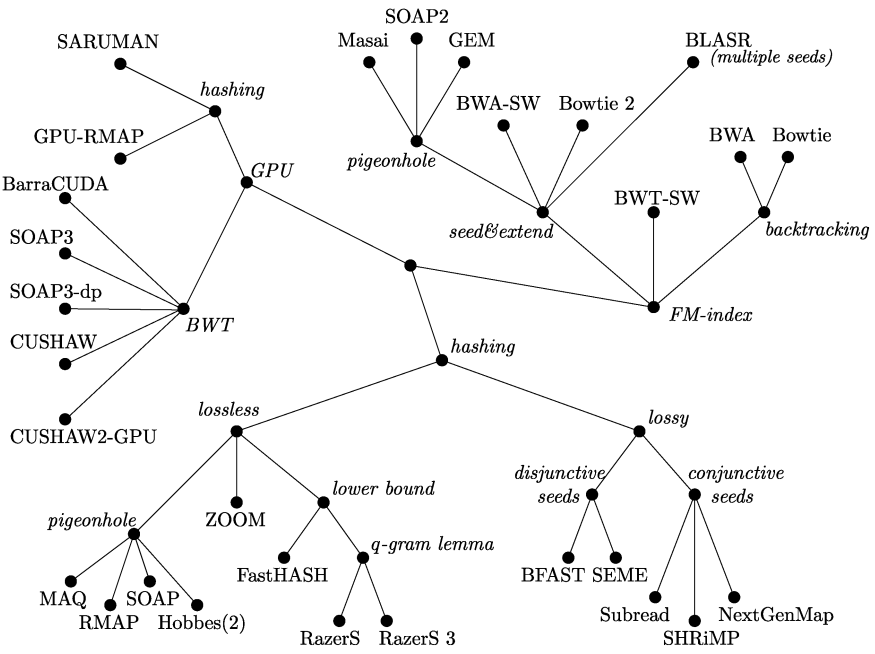

Fig. 16. ***Methods at leaves of a tree that branches along algorithmic design decisions. The hierarchical structure roughly reflects the organization of Sections II–V.***

most computing grids. This explains why the BWT-based FM-index, first implemented in the Bowtie and BWA read mappers, was adopted almost immediately by large numbers of users.

The second prerequisite for the widespread use of mapping software is *accuracy*. On the one hand, the true origin of a read is expected to lie in the set of candidate positions returned by the software. On the other hand, a large number of false positive alignments will essentially hide the true origin from downstream analysis. To map reads with high accuracy, the underlying (error) model must take into account the specifics of the sequencing technology as well as additional data provided by the instrument, such as read pairing information and base quality values. The confidence that the mapper has found the true origin of a read should be reflected by the mapping quality, a valuable piece of information that some downstream analyses rely upon.

Third, the *usability* of the software plays a significant role in its adoption. Typically, scientists who develop and write alignment software and those who run the software have very different backgrounds and skills. Thus, software must be easy to use and well documented, ideally providing some form of user support. For the most popular software packages, user networks have emerged to provide support to one another.

Fourth, *maintenance* is absolutely critical for a package to maintain its usefulness. Although less glamorous than the original development work, maintenance is required to keep up with changes in sequencing technology, changes in underlying operating systems, and the ever-changing ways in which sequence data is used. For example, the leading aligners in 2009 were optimized for read lengths of 35 bp, which was the standard length at the time. This quickly increased to 75 and then 100 bp, and more recently jumped to 300 bp with the new (but lower throughput) MiSeq instrument. Aligners had to modify their seed lengths and other internal methods in order to adjust to these longer reads. At the same time, many users continue to use older, shorter read technology, requiring alignment developers to maintain older versions of their systems simultaneously with the new versions.

As human sequencing work increases, the limitations of using a single reference genome have become apparent. In the near future, we are likely to see additional reference human genomes representing subpopulations or ethnic groups, which in turn may facilitate the analysis of personal genomes [4]. As these become available, it may be useful (see, e.g., [101]) to align reads to multiple genomes simultaneously. This in turn could eliminate the bias towards a single reference and thus improve the accuracy of read mapping. The growing gap between sequencing capacity and computing power, however, needs be filled by clever (mapping) algorithms, that scale sublinearly with genomic data size. The redundancy inherent in collections of genomes can be exploited by storing similarities and variations in a compressed format on which alignment algorithms might operate directly [77].

In the near future, alignment algorithms will not only have to cope with exponentially increasing data volumes,

but also with changing data attributes that might require the development of novel techniques. Most notably, the trend towards longer reads as provided by single-molecule sequencing technologies shifts the DNA read alignment problem slightly towards the problem of aligning whole genomes, which has been studied before. If, on the other hand, longer reads are obtained from RNA transcripts, the mapping algorithms must account for a large number of introns, which might decisively change the nature of the RNA read mapping problem. Furthermore, the higher information content that a longer read carries by itself might allow one to intertwine the read mapping problem and methods that are currently distinct, such as the assembly of full-length transcripts from RNA-seq data. ■

REFERENCES

[1] 1000 Genomes Project Consortium, "A map of human genome variation from population-scale sequencing," *Nature*, vol. 467, no. 7319, pp. 1061–1073, Oct. 2010.

[2] M. I. Abouelhoda and E. Ohlebusch, "A local chaining algorithm and its applications in comparative genomics," in *WABI, Lecture Notes in Computer Science*, vol. 2812, G. Benson and R. D. M. Page, Eds. New York, NY, USA: Springer, 2003, pp. 1–16.

[3] A. Ahmadi *et al.*, "Hobbes: Optimized gram-based methods for efficient read alignment," *Nucl. Acids Res.*, vol. 40, no. 6, p. e41, 2011.

[4] S.-M. Ahn *et al.*, "The first Korean genome sequence and analysis: Full genome sequencing for a socio-ethnic group," *Genome Res.*, vol. 19, no. 9, pp. 1622–1629, 2009.

[5] A. Aji, L. Zhang, and W. Chun Feng, "GPU-RMAP: Accelerating Short-Read Mapping on Graphics Processors," in *Proc. 13th IEEE CSE*, 2010, pp. 168–175.

[6] C. Alkan *et al.*, "Personalized copy number and segmental duplication maps using next-generation sequencing," *Nature Genetics*, vol. 41, no. 10, pp. 1061–1067, Oct. 2009.

[7] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *J. Molecular Biol.*, vol. 215, no. 3, pp. 403–410, Oct. 1990.

[8] S. F. Altschul *et al.*, "Gapped blast and psi-blast: A new generation of protein database search programs," *Nucl. Acids Res.*, vol. 25, no. 17, pp. 3389–3402, Sep. 1997.

[9] A. Apostolico, "The myriad virtues of suffix trees," in *NATO Advance Science Institute Series*, vol. 12. Berlin, Germany: Springer Verlag, 1985, pp. 85–95, ser. F: Computer and Systems Sciences.

[10] R. A. Baeza-Yates and C. H. Perleberg, "Fast and practical approximate string matching," *Inf. Process. Lett.*, vol. 59, no. 1, pp. 21–27, 1996.

[11] J. Blom *et al.*, "Exact and complete short-read alignment to microbial genomes using graphics processing unit programming," *Bioinformatics*, vol. 27, no. 10, pp. 1351–1358, 2011.

[12] A. Blumer *et al.*, "The smallest automaton recognizing the subwords of a text," *Theor. Comput. Sci.*, vol. 40, pp. 31–55, 1985.

[13] R. S. Boyer and J. S. Moore, "A fast string searching algorithm," *Commun. ACM*, vol. 20, no. 10, pp. 762–772, Oct. 1977.

[14] A. Brodnik and J. I. Munro, "Membership in constant time and almost-minimum space," *SIAM J. Comput.*, vol. 28, no. 5, pp. 1627–1640, 1999.

[15] S. Burkhardt *et al.*, "*q*-gram based database searching using a suffix array (QUASAR)," in *Proc. Third Annu. Int. Conf. Comput. Mol. Biol.*, pp. 77–83, 1999.

[16] S. Burkhardt and J. Kärkkäinen, "Better filtering with gapped q-grams," in *CPM, Lecture Notes in Computer Science*, vol. 2089, A. Amir and G. M. Landau, Eds. New York, NY, USA: Springer-Verlag, 2001, pp. 73–85.

[17] M. Burrows and D. J. Wheeler, "A block-sorting lossless data compression algorithm," Digital SRC Res. Rep., Tech. Rep. 124, 1994.

[18] D. Campagna *et al.*, "Pass: A program to align short sequences," *Bioinformatics*, vol. 25, no. 7, pp. 967–968, 2009.

[19] M. Chaisson and G. Tesler, "Mapping single molecule sequencing reads using basic local alignment with successive refinement (BLASR): Theory and application," *BMC Bioinformatics*, vol. 13, no. 238, p. 238, 2012.

[20] S. Chen, A. Wang, and L. M. Li, "SEME: A Fast Mapper of Illumina Sequencing Reads with Statistical Evaluation," *J. Comput. Biol.*, vol. 20, no. 11, pp. 847–860, Nov. 2013.

[21] N. Cloonan *et al.*, "Stem cell transcriptome profiling via massive-scale mRNA sequencing," *Nature Methods*, vol. 5, no. 7, pp. 613–619, Jul. 2008.

[22] M. David, M. Dzamba, D. Lister, L. Ilie, and M. Brudno, "Shrimp2: Sensitive yet practical short read mapping," *Bioinformatics*, vol. 27, no. 7, pp. 1011–1012, 2011.

[23] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[24] A. L. Delcher *et al.*, "Alignment of whole genomes," *Nucl. Acids Res.*, vol. 27, no. 11, pp. 2369–2376, 1999.

[25] M. A. DePristo *et al.*, "A framework for variation discovery and genotyping using next-generation DNA sequencing data," *Nature Genetics*, vol. 43, no. 5, pp. 491–498, May 2011.

[26] R. Doolittle *et al.*, "Simian sarcoma virus onc gene, v-sis, is derived from the gene (or genes) encoding a platelet-derived growth factor," *Science*, vol. 221, no. 4607, pp. 275–277, 1983.

[27] T. A. Down *et al.*, "A Bayesian deconvolution strategy for immunoprecipitation-based DNA methylome analysis," *Nature Biotechnol.*, vol. 26, no. 7, pp. 779–785, Jul. 2008.

[28] D. Eppstein, Z. Galil, R. Giancarlo, and G. F. Italiano, "Sparse dynamic programming i: Linear cost functions," *J. ACM*, vol. 39, no. 3, pp. 519–545, Jul. 1992.

[29] B. Ewing, L. Hillier, M. Wendl, and P. Green, "Base-calling of automated sequencer traces using Phred. I. Accuracy assessment," *Genome Res.*, vol. 8, no. 3, pp. 175–185, Mar. 1998.

[30] M. Farrar, "Striped Smith-Waterman speeds database searches six times over other SIMD implementations," *Bioinformatics*, vol. 23, no. 2, pp. 156–161, Jan. 2007.

[31] P. Ferragina, R. González, G. Navarro, and R. Venturini, "Compressed text indexes: From theory to practice," *J. Exp. Algor.*, vol. 13, pp. 12:1.12–12:1.31, Feb. 2009.

[32] P. Ferragina and G. Manzini, "Opportunistic data structures with applications," in *Proc. 41st Annu. IEEE FOCS*, Washington, DC, USA, 2000, pp. 390–398.

[33] P. Ferragina and G. Manzini, "Indexing compressed text," *J. ACM*, vol. 52, no. 4, pp. 552–581, 2005.

[34] P. Ferragina, G. Manzini, V. Mäkinen, and G. Navarro, "Compressed representations of sequences and full-text indexes," *Trans. Algor.*, vol. 3, no. 2, May 2007, Art. ID. 20.

[35] N. A. Fonseca, J. Rung, A. Brazma, and J. C. Marioni, "Tools for mapping high-throughput sequencing data," *Bioinformatics*, vol. 28, no. 24, pp. 3169–3177, 2012.

[36] P. M. Gontarz, J. Berger, and C. F. Wong, "Srmapper: A fast and sensitive genome-hashing alignment tool," *Bioinformatics*, vol. 29, no. 3, pp. 316–321, 2013.

[37] S. Gräf *et al.*, "Optimized design and assessment of whole genome tiling arrays," *Bioinformatics*, vol. 23, no. 13, pp. i195–i204, 2007.

[38] R. Grossi and J. S. Vitter, "Compressed suffix arrays and suffix trees with applications to text indexing and string matching (extended abstract)," in *Proc. ACM STOC*, 2000, pp. 397–406.

[39] F. Hach *et al.*, "mrsFast: A cache-oblivious algorithm for short-read mapping," *Nature Methods*, vol. 7, no. 8, pp. 576–577, Aug. 2010.

[40] A. Hatem, D. Bozdag, A. Toland, and U. Catalyurek, "Benchmarking short sequence mapping tools," *BMC Bioinformatics*, vol. 14, no. 1, p. 184, 2013.

[41] J. Healy, E. E. Thomas, J. T. Schwartz, and M. Wigler, "Annotating large genomes with exact word matches," *Genome Res.*, vol. 13, no. 10, pp. 2306–2315, Oct. 2003.

[42] N. Homer, B. Merriman, and S. F. Nelson, "BFAST: An Alignment Tool for Large Scale Genome Resequencing," *PLoS ONE*, vol. 4, no. 11, p. e7767, Nov. 2009.

[43] H. Hyyrö, "A bit-vector algorithm for computing Levenshtein and Damerau edit distances," *Nordic J. Comput.*, vol. 10, no. 1, pp. 29–39, Mar. 2003.

[44] H. Jiang and W. H. Wong, "Seqmap: Mapping massive amount of oligonucleotides to the genome," *Bioinformatics*, vol. 24, no. 20, pp. 2395–2396, 2008.

[45] D. S. Johnson, A. Mortazavi, R. M. Myers, and B. Wold, "Genome-wide mapping of *in vivo* protein-dna interactions," *Science*, vol. 316, no. 5830, pp. 1497–1502, 2007.

[46] P. Jokinen and E. Ukkonen, "Two algorithms for approximate string matching in static texts," *Math. Found. Comput. Sci.*, vol. 520, pp. 240–248, 1991.

[47] J. Kärkkäinen and J. C. Na, "Faster filters for approximate string matching," in *Proc. Nine Workshop Algorithm Eng. Experiments (ALENEX)*, New Orleans, LA, USA, Jan. 6, 2007, pp. 84–90.

[48] J. Kärkkäinen and E. Ukkonen, "Lempel-Ziv parsing and sublinear-size index structures for string matching (extended abstract)," in *Proc. 3rd WSP*, 1996, pp. 141–155.

[49] W. J. Kent, "Blatthe blast-like alignment tool," *Genome Res.*, vol. 12, no. 4, pp. 656–664, 2002.

[50] J. Kim, C. Li, and X. Xie, "Improving read mapping using additional prefix grams," *BMC Bioinformatics*, vol. 15, no. 1, p. 42, 2014.

[51] P. Klus *et al.*, "BarraCUDA-a fast short read sequence aligner using graphics processing units," *BMC Res. Notes*, vol. 5, no. 1, p. 27, 2012.

[52] D. E. Knuth, J. H. Morris, and V. R. Pratt, "Fast pattern matching in strings," *SIAM J. Comput.*, vol. 6, no. 2, pp. 323–350, Mar. 1977.

[53] S. Koren *et al.*, "Hybrid error correction and de novo assembly of single-molecule sequencing reads," *Nature Biotechnol.*, vol. 30, no. 7, pp. 693–700, Jul. 2012.

[54] G. Kucherov, L. No, and M. Roytberg, "Multi-seed lossless filtration," in *Combinatorial Pattern Matching, Lecture Notes in Computer Science*, vol. 3109, S. Sahinalp, S. Muthukrishnan, and U. Dogrusoz, Eds. Berlin, Germany: Springer-Verlag, 2004, pp. 297–310.

[55] T. Lam *et al.*, "High throughput short read alignment via bi-directional BWT," in *Proc. IEEE BIBM*, 2009, pp. 31–36.

[56] T. W. Lam, W. K. Sung, S. L. Tam, C. K. Wong, and S. M. Yiu, "Compressed indexing and local alignment of DNA," *Bioinformatics*, vol. 24, no. 6, pp. 791–797, Mar. 2008.

[57] B. Langmead and S. L. Salzberg, "Fast gapped-read alignment with Bowtie 2," *Nature Methods*, vol. 9, no. 4, pp. 357–359, Apr. 2012.

[58] B. Langmead, M. Schatz, J. Lin, M. Pop, and S. Salzberg, "Searching for SNPS with cloud computing," *Genome Biol.*, vol. 10, no. 11, p. R134, 2009.

[59] B. Langmead, C. Trapnell, M. Pop, and S. Salzberg, "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome," *Genome Biology*, vol. 10, no. 3:R25, 2009.

[60] W.-P. Lee *et al.*, "MOSAIK: A hash-based algorithm for accurate next-generation sequencing read mapping," *ArXiv e-prints*, Sep. 2013.

[61] V. Levenshtein, "Binary codes capable of correcting spurious insertions and deletions of ones," *Problems Inf. Transmission*, vol. 1, pp. 8–17, 1965.

[62] H. Li and R. Durbin, "Fast and accurate short read alignment with Burrows-Wheeler transform," *Bioinformatics*, vol. 25, no. 14, pp. 1754–1760, Jul. 2009.

[63] H. Li and R. Durbin, "Fast and accurate long-read alignment with Burrows-Wheeler transform," *Bioinformatics*, vol. 26, no. 5, pp. 589–595, 2010.

[64] H. Li, J. Ruan, and R. Durbin, "Mapping short DNA sequencing reads and calling variants using mapping quality scores," *Genome Res.*, vol. 18, p. 1851, 2008.

[65] M. Li, B. Ma, D. Kisman, and J. Tromp, "Patternhunter II: Highly sensitive and fast homology search," *J. Bioinformatics Comput. Biol.*, vol. 2, no. 3, pp. 417–440, 2004.

[66] R. Li, Y. Li, K. Kristiansen, and J. W. Wang, "SOAP: Short oligonucleotide alignment program," *Bioinformatics*, vol. 24, no. 5, pp. 713–714, 2008.

[67] R. Li *et al.*, "Soap2: An improved ultrafast tool for short read alignment," *Bioinformatics*, vol. 25, no. 15, pp. 1966–1967, 2009.

[68] R. Li *et al.*, "De novo assembly of human genomes with massively parallel short read sequencing," *Genome Res.*, vol. 20, no. 2, pp. 265–272, Feb. 2010.

[69] Y. Liao, G. K. Smyth, and W. Shi, "The subread aligner: Fast, accurate and scalable read mapping by seed-and-vote," *Nucl. Acids Res.*, vol. 41, no. 10, p. e108, 2013.

[70] H. Lin, Z. Zhang, M. Q. Zhang, B. Ma, and M. Li, "Zoom! Zillions of oligos mapped," *Bioinformatics*, vol. 24, no. 21, pp. 2431–2437, 2008.

[71] R. A. Lippert, "Space-efficient whole genome comparisons with Burrows-Wheeler transforms," *J. Comput. Biol.*, vol. 12, no. 4, pp. 407–415, May 2005.

[72] R. Lister *et al.*, "Highly integrated single-base resolution maps of the epigenome in Arabidopsis," *Cell*, vol. 133, no. 3, pp. 523–536, May 2008.

[73] C.-M. Liu *et al.*, "SOAP3: Ultra-fast GPU-based parallel alignment tool for short reads," *Bioinformatics*, vol. 28, no. 6, pp. 878–879, 2012.

[74] Y. Liu, D. Maskell, and B. Schmidt, "CUDASW++: Optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units," *BMC Res. Notes*, vol. 2, no. 1, p. 73, 2009.

[75] Y. Liu and B. Schmidt, "CUSHAW2-GPU: Empowering faster gapped short-read alignment using GPU computing," *IEEE Design Test Comput.*, vol. 31, no. 1, pp. 31–39, Feb. 2014.

[76] Y. Liu, B. Schmidt, and D. Maskell, "CUDASW++2.0: Enhanced Smith-Waterman protein database search on CUDA-enabled GPUs based on SIMT and virtualized SIMD abstractions," *BMC Res. Notes*, vol. 3, no. 1, p. 93, 2010.

[77] Y. Liu, B. Schmidt, and D. L. Maskell, "CUSHAW: A CUDA compatible short read aligner to large genomes based on the Burrows-Wheeler transform," *Bioinformatics*, vol. 28, no. 14, pp. 1830–1837, 2012.

[78] P.-R. Loh, M. Baym, and B. Berger, "Compressive genomics," *Nature Biotechnol.*, vol. 30, no. 7, pp. 627–630, Jul. 2012.

[79] G. Lunter and M. Goodson, "Stampy: A statistical algorithm for sensitive and fast mapping of illumina sequence reads," *Genome Res.*, vol. 21, no. 6, pp. 936–939, 2010.

[80] R. Luo *et al.*, "SOAP3-dp: Fast, Accurate and Sensitive GPU-Based Short Read Aligner," *PLoS ONE*, vol. 8, no. 5, p. e65632, May 2013.

[81] B. Ma, J. Tromp, and M. Li, "Patternhunter: Faster and more sensitive homology search," *Bioinformatics*, vol. 18, no. 3, pp. 440–445, 2002.

[82] V. Mäkinen, N. Välimäki, A. Laaksonen, and R. Katainen, "Unified view of backward backtracking in short read mapping," in *Algorithms and Applications, Lecture Notes in Computer Science*, vol. 6060, T. Elomaa, H. Mannila, and P. Orponen, Eds. Berlin, Germany: Springer-Verlag, 2010, pp. 182–195, ch. 13.

[83] U. Manber and E. W. Myers, "Suffix arrays: A new method for on-line string searches," *SIAM J. Comput.*, vol. 22, no. 5, pp. 935–948, 1993.

[84] S. Marco-Sola, M. Sammeth, R. Guigo, and P. Ribeca, "The GEM mapper: Fast, accurate and versatile alignment by filtration," *Nature Methods*, vol. 9, no. 12, pp. 1185–1188, Dec. 2012.

[85] F. Menges, G. Narzisi, and B. Mishra, "TotalReCaller: Improved accuracy and performance via integrated alignment and base-calling," *Bioinformatics*, vol. 27, no. 17, pp. 2330–2337, 2011.

[86] S. Misra, A. Agrawal, W.-k. Liao, and A. Choudhary, "Anatomy of a hash-based long read sequence mapping algorithm for next generation dna sequencing," *Bioinformatics*, vol. 27, no. 2, pp. 189–195, 2011.

[87] A. Mortazavi, B. A. Williams, K. McCue, L. Schaeffer, and B. Wold, "Mapping and quantifying mammalian transcriptomes by RNA-SEQ," *Nature Methods*, vol. 5, no. 7, pp. 621–628, Jul. 2008.

[88] G. Myers, "A fast bit-vector algorithm for approximate string matching based on dynamic programming," *J. ACM*, vol. 46, no. 3, pp. 395–415, 1999.

[89] G. Narzisi *et al.*, "Accurate de novo and transmitted indel detection in exome-capture data using microassembly," *Nature Methods*, vol. 11, no. 10, pp. 1033–1036, Oct. 2014.

[90] G. Navarro and V. Mäkinen, "Compressed full-text indexes," *Comput. Surv.*, vol. 39, no. 1, Apr. 2007, Art. ID. 2.

[91] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *J. Molecular Biol.*, vol. 48, no. 3, pp. 443–453, 1970.

[92] Z. Ning, A. J. Cox, and J. C. Mullikin, "SSAHA: A fast search method for large DNA databases," *Genome Res.*, vol. 11, no. 10, pp. 1725–1729, Oct. 2001.

[93] W. R. Pearson and D. J. Lipman, "Improved tools for biological sequence comparison," *Proc. Nat. Acad. Sci. USA*, vol. 85, no. 8, pp. 2444–2448, Apr. 1988.

[94] A. M. Phillippy, X. Deng, W. Zhang, and S. L. Salzberg, "Efficient oligonucleotide probe selection for pan-genomic tiling arrays," *BMC Bioinformatics*, vol. 10, p. 293, 2009.

[95] K. R. Rasmussen, J. Stoye, and E. W. Myers, "Efficient q-gram filters for finding all epsilon-matches over a given length," *J. Comput. Biol.*, vol. 13, no. 2, pp. 296–308, 2006.

[96] G. Rizk and D. Lavenier, "GASSST: Global alignment short sequence search tool," *Bioinformatics*, vol. 26, no. 20, pp. 2534–2540, 2010.

[97] S. M. Rumble *et al.*, "SHRiMP: Accurate Mapping of Short Color-space Reads," *PLoS Comput Biol*, vol. 5, no. 5, p. e1000386+, May 2009.

[98] D. Sankoff and J. B. Kruskal, *Time Warps, String Edits, Macromolecules: The Theory and Practice of Sequence Comparison*. Reading, MA, USA: Addison-Wesley, 1983.

[99] M. Schatz, A. Delcher, and S. Salzberg, "Assembly of large genomes using second-generation sequencing," *Genome Res.*, vol. 20, no. 9, pp. 1165–1173, 2010.

[100] M. C. Schatz, "CloudBurst: Highly sensitive read mapping with MapReduce," *Bioinformatics*, vol. 25, no. 11, pp. 1363–1369, 2009.

[101] K. Schneeberger *et al.*, "Simultaneous alignment of short reads against multiple genomes," *Genome Biol.*, vol. 10, no. 9, p. R98, 2009.

[102] S. Schwartz *et al.*, "Human-mouse alignments with BLASTZ," *Genome Res.*, vol. 13, no. 1, pp. 103–107, 2003.

[103] F. J. Sedlazeck, P. Rescheneder, and A. von Haeseler, "NextGenMap: Fast and accurate read mapping in highly

polymorphic genomes," *Bioinformatics*, vol. 29, no. 21, pp. 2790–2791, 2013.

[104] J. T. Simpson and R. Durbin, "Efficient de novo assembly of large genomes using compressed data structures," *Genome Res.*, vol. 22, no. 3, pp. 549–556, Mar. 2012.

[105] E. Siragusa, D. Weese, and K. Reinert, "Fast and accurate read mapping with approximate seeds and multiple backtracking," *Nucl. Acids Res.*, vol. 41, no. 7, p. e78, 2013.

[106] A. Smith, Z. Xuan, and M. Zhang, "Using quality scores and longer reads improves accuracy of Solexa read mapping," *BMC Bioinformatics*, vol. 9, no. 1, p. 128+, Feb. 2008.

[107] A. D. Smith *et al.*, "Updates to the RMAP short-read mapping software," *Bioinformatics*, vol. 25, no. 21, pp. 2841–2842, 2009.

[108] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *J. Molecular Biol.*, vol. 147, no. 1, pp. 195–197, Mar. 1981.

[109] Y. Sun and J. Buhler, "Designing multiple simultaneous seeds for DNA similarity search," *J. Comput. Biol.*, vol. 12, no. 6, pp. 847–861, 2005.

[110] S. Tuupanen *et al.*, "The common colorectal cancer predisposition snp rs6983267 at chromosome 8q24 confers potential to enhanced WNT signaling," *Nature Genetics*, vol. 41, no. 8, pp. 885–890, Aug. 2009.

[111] E. Ukkonen, "Approximate string-matching over suffix trees," in *CPM, Lecture Notes in Computer Science*, vol. 684, A. Apostolico, M. Crochemore, Z. Galil, and U. Manber, Eds. New York, NY, USA: Springer, 1993, pp. 228–242.

[112] W. Wang, P. Zhang, and X. Liu, "Short read DNA fragment anchoring algorithm," *BMC Bioinformatics*, vol. 10, no. S-1, p. S17+, 2009.

[113] H. S. Warren, *Hacker's Delight*. Boston, MA, USA: Addison-Wesley Longman, 2002.

[114] D. Weese, A.-K. Emde, T. Rausch, A. Dring, and K. Reinert, "Razersfast read mapping with sensitivity control," *Genome Res.*, vol. 19, no. 9, pp. 1646–1654, 2009.

[115] D. Weese, M. Holtgrewe, and K. Reinert, "Razers 3: Faster, fully sensitive read mapping," *Bioinformatics*, vol. 28, no. 20, pp. 2592–2599, 2012.

[116] W. J. Wilbur and D. J. Lipman, "Rapid similarity searches of nucleic acid and protein data banks," *Proc. Nat. Acad. Sci. USA*, vol. 80, no. 3, pp. 726–730, Feb. 1983.

[117] R. Wilton *et al.*, "Arioc: High-throughput read alignment with GPU-accelerated exploration of the seed-and-extend search space," *PeerJ*, vol. 3, p. e808, 2015.

[118] H. Xin *et al.*, "Accelerating read mapping with fasthash," *BMC Genomics*, vol. 14, no. S-1, p. S13, 2013.

ABOUT THE AUTHORS

Stefan Canzar received that Diploma in computer science from the Technische Universität München, Munich, Germany, in 2004, and the bi-national doctoral degree in computer science from Universität des Saarlandes, Saarbrücken, Germany, and Université Henri Poincaré, Nancy, France, in 2008.

From 2009 to 2011 and from 2012 to 2014, he was a Postdoctoral Researcher with the Centrum Wiskunde & Informatica, Amsterdam, The Netherlands, and the Johns Hopkins Institute of Genetic Medicine, Baltimore, MD, USA, respectively. Since 2014, he has been a Research Assistant Professor with the Toyota Technological Institute, Chicago, IL, USA. His research interests include the development of algorithmic solutions to problems arising in the analysis of high-throughput sequencing data. The goal of his research is to develop advanced computational approaches that help to transform the generated data into information and ultimately knowledge in research and medicine.

Steven L. Salzberg received the B.A. degree in English and M.S. and M.Phil. degrees in computer science from Yale University, New Haven, CT, USA, in 1980, 1982, and 1984, respectively, and the Ph.D. degree in computer science from Harvard University, Cambridge, MA, USA, in 1989.

From 1989 to 1998, he was a faculty member in computer science with Johns Hopkins University, Baltimore, MD, USA, and from 1997 to 2005, he was Senior Director of Bioinformatics with The Institute for Genomic Research (TIGR), Rockville, MD, one of the world's leading DNA sequencing centers at the time. From 2005 to 2011, he was the Director of the Center for Bioinformatics and Computational Biology (CBCB) and the Horvitz Professor of Computer Science with the University of Maryland, College Park, MD. He is currently the Bloomberg Distinguished Professor of Biomedical Engineering, Computer Science, and Biostatistics and the Director of the Center for Computational Biology with the McKusick-Nathans Institute of Genetic Medicine, Johns Hopkins University. He has authored or coauthored over 250 publications in leading scientific journals, and his h-index is 113.

Dr. Salzberg is a Fellow of the American Association for the Advancement of Science, a Fellow of the International Society for Computational Biology, and a former member of the Board of Scientific Counselors of the National Center for Biotechnology Information at NIH. He was the 2013 winner of the Benjamin Franklin Award for Open Access in the Life Sciences, and the 2013 winner of the Robert G. Balles Prize in Critical Thinking for his Forbes science column. In 2001 and again in 2014, he was listed as a Highly Cited Researcher by Thomson Reuters, a compilation of the 1% most-cited researchers in the world.