Published in final edited form as:
Nat Rev Genet. 2023 April ; 24(4): 235–250. doi:10.1038/s41576-022-00551-z.

Navigating bottlenecks and trade-offs in genomic data analysis

Bonnie Berger[1,2,5,✉], **Yun William Yu**[3,4,5]

[1]Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA, USA.

[2]Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA.

[3]Department of Computer and Mathematical Sciences, University of Toronto Scarborough, Toronto, Ontario, Canada.

[4]Tri-Campus Department of Mathematics, University of Toronto, Toronto, Ontario, Canada.

[5]These authors contributed equally: Bonnie Berger and Yun William Yu.

Abstract

Genome sequencing and analysis allow researchers to decode the functional information hidden in DNA sequences as well as to study cell to cell variation within a cell population. Traditionally, the primary bottleneck in genomic analysis pipelines has been the sequencing itself, which has been much more expensive than the computational analyses that follow. However, an important consequence of the continued drive to expand the throughput of sequencing platforms at lower cost is that often the analytical pipelines are struggling to keep up with the sheer amount of raw data produced. Computational cost and efficiency have thus become of ever increasing importance. Recent methodological advances, such as data sketching, accelerators and domain-specific libraries/languages, promise to address these modern computational challenges. However, despite being more efficient, these innovations come with a new set of trade-offs, both expected, such as accuracy versus memory and expense versus time, and more subtle, including the human expertise needed to use non-standard programming interfaces and set up complex infrastructure. In this Review, we discuss how to navigate these new methodological advances and their trade-offs.

Introduction

Historically, genomic sequencing has been much more expensive and time-consuming than the computational analyses that follow. If sequencing costs were tens of thousands of dollars for a single sample and the per-sample compute was only a few dollars, then compute costs became an afterthought. Researchers had little need to think about allocating resources for computation, or about what types of analyses would be faster and, thus, more cost-effective. Obviously, faster algorithms were always preferable, but if a 10 times slower algorithm

Reprints and permissions information is available at www.nature.com/reprints.

✉**Correspondence** should be addressed to Bonnie Berger. bab@mit.edu.

The authors contributed equally to all aspects of the article.

Competing interests
The authors declare no competing interests.

was only 5% more accurate, then it often made sense to use the slower, more accurate algorithm because so little of the overall cost of a sequencing analysis was bound up in compute times. Compute hardware was even less of a choice for individual experiments, as decisions were made on a `technology refresh lifecycle`; it would be atypical to buy a new compute server for a single analysis. Thus, it was generally unnecessary to constantly consider trade-offs between different performance characteristics.

However, the continued drive to bring down the cost of sequencing and expand the throughput of sequencing platforms has proven successful[1] — Illumina short-read technology can sequence a full genome for around US$600 (ref.[2]), and Ultima Genomics and Beijing Genomics Institute (BGI) report that they can sequence a genome for around US$100 (refs.[3,4]). Advances in computing have not kept up, so we cannot simply rely on `Moore’s law`, which relates to `time-complexity`, or `Kryder’s law`, which relates to `space-complexity`, to make computation commensurately cheaper[5]. Analytical pipelines are often overwhelmed in processing the massive data that reside in different laboratories, companies and biobanks. Coupled with the explosion of raw data from single-cell sequencing[6,7], as well as studies that employ the large-scale re-analysis of existing publicly available data[8], computation is becoming a considerable part of the total cost. As such, scientists who use or process these data not only must plan on acquiring sufficient `compute resources` but must now also consider trading off accuracy, compute, storage and complexity of infrastructure.

Recent computational advances in algorithms, hardware and programming languages have emerged that account for some of these trade-offs and other more subtle trade-offs. Data `sketching`, akin to a sketch of a scene by an artist, can provide orders of magnitude speed-up by using ‘lossy’ approximations that give up perfect fidelity to capture only the most important features of the original[9]; this can allow researchers to continue thinking of computational cost as negligible, but only if they are willing to give up the possibility of perfect accuracy. `Accelerators` (such as `field-programmable gate arrays` (FPGAs) and `graphics processing units` (GPUs))[10] and cutting-edge `domain-specific languages`[11] may require additional investment either in expensive hardware or training of users; yet accelerators provide significant speed-ups, and domain-specific languages can make it easier for programmers to reproducibly and consistently handle complex operations that arise often in practice. Furthermore, the ease with which it is now possible to outsource compute to the cloud has turned hardware decisions from mattering primarily during technology refresh cycles of sequencing cores to choices and options present during every single experimental analysis. In the past, a scientist faced with a week-long wait for an analysis to complete using local resources simply had to wait; now, they typically have the option to run it on the cloud, renting possibly specialized hardware, but with a new set of costs. It is thus not just computational scientists who have to understand these approaches but also the biological and biomedical researchers who use them; the decision of what trade-offs are worthwhile is one of overall experimental design and resource allocation.

As an example, suppose a clinician wants to analyse a patient’s genome and gut microbiome for the purposes of personalizing medical treatment. One approach could be to fully sequence, map and call variants for both the human and bacterial genomes using the best practices of the genome analysis toolkit (GATK)[12]. On a 30× coverage Illumina short-read sample, the full pipeline might take upwards of 10 h on a typical compute server[13]. Another choice could be to process it in the cloud (for example, via Google Cloud Platform) for US$5 worth of compute, but it would still take tens of hours and the data also need to be sent to the cloud[14]. However, a functionally equivalent analysis using Illumina Dragen hardware acceleration could be completed in less than an hour but would cost $20 to outsource to Amazon Web Services[15]. Alternatively, a more targeted analysis might be all that is required by the clinician: perhaps it would be sufficient to look for specific marker genes using an alignment-free computational SNP array[16] and comparing the microbiome at a `k-mer` distribution level using Mash with other patients[17], but this may give up accuracy for speed. There are now many available options for practitioners to decide between, and with the advent of `cloud computing` they may freely make a different choice for each analysis, depending on both the local compute resources they possess and the speed and accuracy they need.

In this Review, we discuss how to navigate new and emerging technologies for handling the computational nature of secondary processing as a bottleneck in the wider pipeline of genomic sequencing analyses (Fig. 1). Following primary generation of the sequencing data (Box 1), secondary processing refers to diverse types of analyses, including computing genomic variation and assembling or mapping of raw reads (Box 2); this step precedes tertiary processing to answer specific biological questions, which invokes methods such as genome annotation[18], genome-wide association studies (GWAS)[19], machine learning predictions/classifications[20] and so on. The Review will not cover important topics related to non-traditional sequencing data such as proteomic[21], epigenetic[22] and spatial modalities[23,24], as well as privacy of genomic data[25,26].

We start by discussing how analytical bottlenecks create trade-offs in genomic analyses. Algorithm designers generally are concerned with technical trade-offs in compute, memory and communication, but those give rise to user-facing trade-offs of time, money and complexity of deployment. We then discuss newer methods, which sometimes allow for a continuous trade-off of two or more desirable characteristics, a decision that increasingly falls on the end users. As background, we also give a short overview of newer data acquisition technologies that have recently emerged, including long-read sequencing, single-cell sequencing and more (Box 1, Table 1). These technologies are not only adding to the data deluge already present from second-generation short-read sequencing but also introduce different trade-offs in analyses. As the various analytical and hardware advances for addressing the processing bottlenecks are not specific to a single technology, our Review does not focus on the differences in data acquisition and primary processing. Rather, the Review focuses on those various techniques for secondary processing, including data compression and sketching, hardware accelerators, cloud parallelization and domain-specific languages. We conclude by highlighting exciting future directions for the field.

Trade-offs in genomic data analysis

As the cost of primary sequencing decreases, the relative cost of compute in secondary processing increases and is fast becoming an increasingly important bottleneck. However, there are several different types of salient trade-offs; we will be dividing our exposition of the trade-offs into those that are largely technical, such as computation, communication, storage and memory, and those that are more user-facing, including the overall time, expense, complexity of deployment and accuracy of the analysis. Obviously, the two types of trade-offs are related, and understanding the user-facing experience requires delving into the technical considerations.

Technical trade-offs

Compute.—Computational cost is one of the first things people think about when they think of algorithms and processing; that is, the number of central processing unit (CPU) operations an algorithm uses, which is often a rough proxy for the amount of time it takes. Classically, in the algorithms literature[27], computer scientists have largely focused on the `complexity` class of algorithms, which refers to how the algorithms `scale` with the amount of data to be analysed. However, the raw amount of genomics data is huge — with, for example, The Cancer Genome Atlas (TCGA)[28,29] data, the files are massive (often 50–100 GB compressed), and even just `parsing` them takes a lot of time. Thus, in order to even be in contention, most practical algorithms have an asymptotic time-complexity that scales either linearly or almost linearly (log-linearly) with the size of the data. Of course, algorithms are still faster or slower because of multiplicative constant factors; for example, all else equal, an algorithm that has to read all of the data twice will be slower than one that reads through the data only once. Although in this Review we focus on the big-ticket items of mapping and variant calling (Box 2), there are also many smaller tasks, such as sorting reads, quality control, fixing `CIGAR strings` and calculating quality score profiles. None of these are independently as ‘heavy’ as mapping or genotype calling, and they are all at worst log-linear time, but the multiple independent steps add up and can sometimes even dwarf the time needed to map the reads or call the genotypes. Scaling behaviour is important, but in practice actual speeds may vary.

Space.—Space costs — that is, storage and memory — are usually the second type of technical cost people consider. Although magnetic tape can be obtained for less than one US cent per gigabyte[30] and more-managed cloud-based options such as Amazon Glacier Deep Archive are priced at approximately one cent per gigabyte-year, storage — unlike compute — is an ongoing cost and cumulative over time. Data must be maintained indefinitely, not only to support reproducibility of individual studies but also to enable later re-analysis using more recent tools and to allow larger meta-studies that analyse data from many studies. For example, the UK Biobank has hundreds of thousands of samples[31], and that number will only grow with time. Furthermore, with the decrease in the cost of sequencing, it is possible to run experiments studying questions that require more data; indeed, modern genomics projects often use data from thousands of samples. An individual sample may be stored in compressed form in tens of gigabytes, but that scales up to many terabytes for thousands of samples and to petabytes for entire biobanks. New approaches such as storing data long-term in genomic samples themselves are exciting science[32], but for the foreseeable future we still need to use conventional mechanisms for storage.

Even more expensive on a per-byte level is shorter-term storage[33]. Any sequencing data that researchers expect to analyse in the near-term lives on either hard disk drives (HDDs), solid-state drives (SSDs) or the cloud. Both genomics-specific compression formats (for example, CRAM[34] and BAM[35]) and general-purpose compression (gzip-compressed FASTQ[36] files) are in widespread use[37]. However, storing in compressed format usually requires additional processing in the form of decompression before running analyses, so compression algorithms allow trading off storage space with computational cost. Some analysis methods can operate directly on compressed data, a technique known as ‘compressive genomics’[38], which reduces or sometimes entirely eliminates that trade-off, but these are the exception rather than the rule. As we move from long-term to short-term storage, we eventually get to in-memory storage (`random access memory` (RAM)), which is often considered a hard constraint on algorithms. To a first-order approximation, methods that require too much RAM are simply not runnable on low-RAM machines, because the algorithm requires fast access to all of the data in RAM; however, higher RAM machines are rapidly coming down in price. In the past, users who ran into a RAM bottleneck might have needed to use different analysis software or wait to acquire a higher RAM machine; now, a viable alternative solution would be to instead provision a high-RAM cloud instance on demand.

Communication.—Because different forms of storage have their pros and cons, communication costs between both different forms of storage and different parties become important. This is in many ways a much more severe bottleneck than storage itself, because it directly affects the time it takes to perform an analysis in settings where the data-gathering party and compute party are different, such as in cloud computing[39]. Communication costs have practical consequences for practitioners because they need to decide on the trade-offs of where to perform certain computations — do they send only the outputs of analysis, or rather the original data and allow the recipient to compute? For example, sometimes sending a raw compressed FASTQ sequencing file to a central compute server might be smaller than sending a mapped sequence alignment map (SAM)/BAM file, but that then puts all the computational burden on the central server. On the other hand, sending a variant call format (VCF)[40] file uses much less space than sending either raw FASTQ or SAM/BAM files, but places nearly all the computational burden on the sending party; moreover, this means that a central party doing a large analysis on samples from multiple parties does not have access to the raw data, which may limit the types of analyses they can do. VCF files may allow for performing GWAS analyses, but they do not allow re-phasing of the data; that is, re-identification of which allele is inherited from which chromosome (Box 2).

Communication bottlenecks have become especially important as single-cell sequencing takes off[41]. Single-cell sequencing is currently still more expensive to generate per base because of library preparation costs, but the total raw reads for an individual can be greater than for normal bulk sequencing because there are distinct reads from each cell in the study. Thus, often practitioners will not send around the raw reads but, instead, precompute a (sometimes normalized) count matrix of expressed genes per cell[42,43]. Unfortunately, doing this removes a large amount of the information from the raw reads, can skew results and can make it harder to perform non-standard analysis[44]. This means that the trade-off is not only in technical variables such as communication, computation and memory but also bleeds into the science an end user is able to do.

These communication costs are increasingly critical as cloud computing becomes more popular. When data are stored, for example, in the Google Cloud, egress costs of about 12 cents per gigabyte[45] can dominate storage and compute costs. On the other hand, as we fully enter the era of cloud-based biobanks[31,46–49], a popular solution is simply to never perform egress. Cloud-based biobanks are large-scale biomedical databases available for research use on cloud platforms, where the expectation is that the raw data and compute all stay in the cloud, and only the resulting analysis is egressed — the UK Biobank[31] and the US National Institutes of Health (NIH) All of Us[50] projects have each generated whole-genome sequencing data for more than 100,000 individuals and counting. Although these resources provide a treasure trove of data for population and medical genetics studies, sequencing files cannot be egressed from and can be analysed only on cloud-based computing platforms themselves, such as DNAnexus for the UK Biobank. In this scenario, the compute for some large-scale genetic analyses can cost upwards of US$1 million. Furthermore, these cloud platforms often require platform-specific expertise; thus, research is far from democratized.

User-facing bottlenecks

Although the technical details of trade-offs are of primary importance to method developers, they often also affect the user experience. When the major bottlenecks in genomics pipelines were related to data collection and sequencing, the computational trade-offs could be largely overcome by just allocating more resources to the problem. However, this ‘solution’ only works for the few well-funded laboratories with resources to spare. As we seek to give more end users access to genomic data analysis, this kind of solution is not sufficient. For those users, here we connect the technical trade-offs to the user-facing bottlenecks.

Time.—The most obvious user-facing bottleneck is measured in time. How many minutes/hours/days does it take for the user to receive the results of an analysis they wish to run? Although, classically, algorithm developers will talk about ‘time-complexity’ as a proxy for computation, we separate the two in this Review because the actual time a user has to wait is dependent on many other factors. The number of compute operations needed for the analysis, that is time-complexity, obviously plays a major role, but with increasing `parallelization` and hardware accelerators (FPGAs, GPUs and `tensor processing units` (TPUs)), for those who have access to those kinds of resources the actual wait time for an analysis can be decreased substantially.

However, there are very often many kinds of non-compute bottlenecks that increase the analysis time. Compute clusters are generally heterogeneous[51], with a mix of different types of machines — for example, there may be many machines in a cluster, but only a limited number with state-of-the-art GPUs, large amounts of RAM or many compute cores. As discussed in the ‘Compute’ section above, many bioinformatics pipelines have both more-intense steps (for example, mapping or assembly) and less-intense steps (such as sorting or changing file formats). The more-intense steps often require those limited higher performance machines, so there is often a cluster bottleneck of everyone waiting on the better machines. In addition to queuing for better machines, data transmission, network latency and generally moving data up and down the storage hierarchy can also impose considerable wait times.

The types of acceptable solutions can also vary tremendously depending on the use case. In a research laboratory, the time it takes for a single sample might not be so important, because the key bottleneck is often the time it takes to analyse an entire data set of hundreds of samples, and different samples can often be analysed independently. A `single-threaded` algorithm that takes a week to analyse a single sample could be perfectly acceptable in this scenario, because there may be enough resources to parallelize and analyse thousands of samples in that week. On the other hand, for clinical and personalized genomics, the speed of analysis of each individual sample can be critical, because a clinician may be waiting to make a decision about a particular patient based on the results[52]. Here, modern techniques using accelerators (for example, Illumina Dragen) may allow processing a single sample in the order of minutes instead of hours because they make it possible to parallelize the analysis of a single sample. For example, the GATK best-practices pipeline takes tens of hours for a single whole-genome sequencing 30× coverage sample on the Google Cloud Platform using single-threaded processes, but can be accelerated to less than an hour using Illumina Dragen. Thus, although there is overlap in the bottle necks for research and clinical genomics, they are not the same; research studies can apply parallelism at the sample level without much disadvantage, whereas clinical applications that need fast turnarounds should use some of the new hardware acceleration solutions.

Money.—In the discussion about ways to reduce analysis times, one major recourse is to parallelize across additional computers. However, this manifests in monetary cost, giving rise to the ever-present time–money trade-offs. By purchasing more computers, more cloud nodes or hardware accelerators, users can decrease wait times for individual results. Indeed, in the age of cloud computing, the time–money trade-off is often made explicit, as each limited resource has a specific associated cost, and users may choose to buy extra 'priority' on faster machines for an analysis that they really need to have done now or to pay less for pre-emptible instances that do not guarantee the analysis will finish in a specific amount of time[53]. On the other hand, for computing cores that actually purchase machines in-house, the time–money trade-off is not as continuous but, instead, can be somewhat discontinuous whenever you need more resources than you have. Incorrectly assessing the amount and type of storage needed for an algorithm can greatly raise both the financial and computational cost of analysis — if the computational analysis you are relying on requires 512 GB of RAM and you only have a 256 GB machine, then either you have to use slow disc swap, which can take orders of magnitude more time, or you have to buy more RAM or a different compute machine if your existing hardware does not support that amount of RAM. Unlike with cloud computing, you cannot simply temporarily upgrade to a larger machine[54].

Because the cost of incorrectly predicting resources needed is much higher when purchasing machines, provisioning cloud resources on demand is therefore more forgiving for most users. Alternatively, users (or sequencing cores) who repeatedly run the same analysis pipelines and therefore have predictable resource requirements may find it sensible to purchase in-house machines for those pipelines, knowing that whenever they need to run other pipelines, they always have the option of buying on-demand compute if existing machines are not suitable.

Accuracy.—In addition to the two well-known user-facing trade-offs of expense and time, in this Review we also focus on two less obvious constraints: accuracy and deployment complexity. Although accuracy is arguably a technical trade-off related to the analysis algorithms used, modern sampling-based and sketching-based techniques explicitly can reduce accuracy to different degrees for the benefit of using less computation and communication[55]. Computational biologists have always been implicitly making this trade-off; even classical methods such as BLAST[56] for local alignment are much faster than running a full Smith–Waterman[57] dynamic programming method, but at the cost of lower accuracy[58]. This trade-off also appears in terms of communication: more recently, in single-cell sequencing, just sending the normalized precomputed gene by cell count matrix is much faster, but different methods for computing that matrix demonstrate different accuracies[59,60], so the cell count matrix cannot be treated as a fundamental unit of measurement without care[61]. However, we must be careful not to conflate the accuracy of one step of an analysis pipeline with the validity of conclusions drawn from the pipeline as a whole. Because of inherent noise, genomic analyses are designed to be robust, and we are simply looking for functional equivalence[62] of the overall results. That provides room for trading away a small amount of accuracy.

Still, in the past this trade-off was often hidden behind large jumps in the accuracy–performance continuum, or behind ambiguities in the optimization goals. More accurate methods such as Smith–Waterman are simply too slow to practically use in many instances, so everyone is forced to use the faster alternative of BLAST, or its more modern replacements (for example, BLAT[63] or caBLAST[38]). When the choice is between two different algorithms, it is largely all or nothing. Methods developers can of course design methods with somewhat different trade-offs, such as BLAT, which is faster but less accurate than BLAST[63], but there is no obvious user-facing way to interpolate between algorithms. Furthermore, there is the additional question about whether the scoring model we use to judge accuracy makes sense biologically. Smith–Waterman depends on the substitution matrix and gap penalty scheme chosen, and although it optimally computes local alignments given a particular scheme, this may not always correspond to the most biologically meaningful hits. Indeed, as we will see later when we talk about lossy compression, perfectly accurate reproduction of scores does not always translate to ‘accuracy’ in downstream analysis.

Modern data sketching methods often allow explicit choice of how much accuracy to give up for speed and memory. Thus, it now becomes a parameter choice on which users need to explicitly decide. Is it worth doubling the amount of computation or memory needed to buy yourself a 41% improvement in accuracy? Does that improvement in accuracy for a step in an analysis pipeline even change the final results of the analysis? At what point do you stop? This is not a hypothetical question any longer, because software such as Mash[17] for comparing genomic samples requires users to set parameters controlling exactly that. Unfortunately, because every analysis has different requirements for overall accuracy, and the impact of accuracy of different steps in a pipeline differs, we are unable to give specific numbers as guidance. A useful rule of thumb is that only rough equivalence of the accuracy on the final results of a pipeline matters; it is reasonable to reduce the accuracy of intermediate computations up until the point where it affects the final results more than re-running the experiment would. However, knowing that threshold requires thorough benchmarking of analysis pipelines with different parameter choices in intermediate steps. Nevertheless, at the end of the day, it is incumbent on the end user to make a specific choice of what trade-off they wish to make.

Complexity.—An even more subtle trade-off comes in the form of the complexity of development and deployment of software and infrastructure. The growth of biological data (and in particular sequencing data) has spurred much research into high-performance methods, algorithms and tools for analysing genomic data. For each individual application, the corresponding tools take software engineers months to develop, typically in a low-level language such as C or C++ to optimize performance (for example, BWA-MEM[64], Bowtie2 (ref.[65]) and Minimap2 (ref.[66])). This is especially true for any development of hardware-accelerated software. Many bioinformatics practitioners simply take those high-performance methods, use them as building blocks in pipelines and write their own code to ‘glue’ them together into pipelines, typically in a higher level, much less performant language such as Python[67]. The glue code normally just consists of taking the output of one tool and entering it as input for another tool. Normally, the glue is much less complex and therefore takes negligible amounts of time to run compared with the main methods. However, when new software is developed that uses a different input or output format, the glue code is often tasked with actually converting the file formats. It is still less complex than the main analysis code, but especially as more engineering hours go into the high-performance methods, the glue can take increasingly substantial portions of the total runtime.

For emerging areas, such as in single-cell sequencing, there is not yet much standardization of file formats or nomenclature across the various consortia (for example, Human Cell Atlas[6]), because scientists have not yet figured out what is most salient. Analysing a data integration method (for example, Scanorama[42], Harmony[68] or Seurat[69]) over various single-cell data sets requires conversion to the same format. Thus, getting different pipelines to work together is highly non-trivial and often involves converting files from one format to another. As users are confronted with additional trade-offs and parameter decisions, building interoperable pipelines becomes even more difficult. When there are only specific points in the parameter space that are reasonable — such as when the choice is between just normalized cell counts and the raw FASTQ files for single-cell sequencing — it is relatively straightforward to build tools that take as input one or the other. However, when there is an entire continuum of possible trade-offs, downstream pipelines then must be aware of the many more possibilities. One solution is of course standardization, which will play an important role in generating interoperable file formats[70], but standardization of file formats cannot resolve the issue when the underlying analyses can be performed with an entire range of parameter choices affecting accuracy. Furthermore, the process of setting up an environment to run high-performance bioinformatics software is itself a specialized skill. There are many substantial algorithmic advances that see very little adoption in the community because the software is hard to use or is not well maintained. For instance, we had to replace our compressive read-mapping accelerator (CORA) program[71] with CORA-seq[72]. Substantial effort is required to actually make important methodological advances usable by end users.

Modern computational methodologies

In talking about modern methodologies (Table 2), there are three major themes: algorithmic advances, including data compression and data sketching (Fig. 2); hardware platforms, including accelerators and cloud computing; and development tools, including domain-specific libraries and languages.

Data compression

When one thinks of trade-offs in computational bottlenecks, classically one gives the example of data compression. Compression typically takes advantage of redundancies in the original data to generate compressed encodings that take up fewer bytes than the original. This encoding reduces communication and storage requirements, but generally increases the computational cost because it is difficult (although certainly not impossible) to design algorithms that operate directly on compressed data[73], and decompressing data takes some amount of compute.

There has been a lot of work on genomic data compression over the years[37], some of which is towards standardization (for example, GA4GH (ref.[74]) and MPEG-G[75]). These developments range from simply applying general-purpose compressors (such as Gzip), to applying general-purpose compressors in a way that preserves some useful properties such as `random access`, to fully specialized genomics-specific compression algorithms[76–78]. Most sequencing cores currently use CRAM or BAM formats for mapping the compressed format to a reference genome, both of which preserve relatively straightforward decompression and random access. However, the academic literature includes several more sophisticated compression strategies that are nicely compatible with accelerated bioinformatics algorithms without decompression[5]. For example, reordering or clustering reads before compression[71,79–81] not only can increase compress ratios but also can be used to accelerate similarity search. However, the downstream algorithms must be designed as compression-aware so they can directly read from the compressed file format, and furthermore, the way the compressed file is laid out also limits the types of possible fast queries. A simple example is that when compressing a data matrix, one has to choose whether to organize it by column, by row or by blocks. If it is organized by row, determining the average value in a column is expensive because the entire matrix must be decompressed. Similarly, when compressing genomes for accelerated similarity search[38], it is fast to determine similar genomes, but without additional work may be slow to determine whether every genome contains a particular sequence; of course, that query could be enabled with a different specialized auxiliary data structure[82–84].

These methods tend towards `lossless compression`, or at least nearly lossless. (Strictly speaking, reordering or clustering reads drops ordering information, but that information is generally considered to be an artefact of random chance in the sequencing biology and chemistry.) However, when targeting specific genomics analyses, one can often achieve much higher compression ratios by discarding information. This type of `lossy compression` is in fact standard practice in the realms of audio, image and video compression[85]. Human eyes and ears are largely unable to perceive certain high-frequency patterns, and so those can be safely discarded in JPEG or MP3 files. In the genomics sequencing literature, some of the most prominent examples of lossy compression came from the quality score compression literature, whereby it was found that the self-reported base-calling confidence scores of Illumina sequencers could be safely discarded while not deteriorating, and often even improving, downstream variant calling[75,86–88]. When the downstream pipelines are standardized, it is feasible, therefore, to analyse the effects of discarding information in a systematic manner. Of course, what information can be safely discarded evolves over time as downstream analyses change. Additionally, scientists often have a greater emotional attachment to certain types of data; for example, the primary data of the DNA sequence itself seems somehow more important than metadata, such as quality scores. On the other hand, biologists are already comfortable with that trade-off when it is a limitation of experimental methodology — whole-exome sequencing discards the vast majority of non-coding sequences of course. In both cases, it is crucial to consider the end goal of the data being collected. When sending data to a collaborator for a specific analysis, the operating parameters are quite different than when data are stored for long-term archival purposes.

Data sketching

With lossy compression, the primary goal is to identify information that is not necessary for the related downstream analyses, and thus can be discarded. However, when speaking about lossy compression, we are usually targeting a pre-existing downstream analysis pipeline, and it is important to be able to reconstruct data in the same format as the original data, because that is what the downstream analysis pipeline expects. On the other hand, if we are given the additional degree of freedom of changing the algorithms used for the downstream analysis to accept sketched input, we do not need to be able to reconstruct even a lossy version of the original. This freedom can give rise to even more efficient encodings. This is the principle behind data sketching[9]. Intuitively, lossy compression creates a modified version of the same data that takes up less space, often with the goal of ensuring that biologically relevant features are preserved, whereas data sketching simply extracts those biologically relevant features directly. Data sketching can be more efficient than lossy compression, but it requires the downstream analysis to understand the sketches, whereas lossy compression can work with off-the-shelf tools.

One of the prominent examples of data sketching in the bioinformatics literature is the Mash software[17]. Mash is based on the even older MinHash data sketching algorithm[89,90], which quickly estimates the `Jaccard index`, commonly used for measuring the similarity[91]. The premise behind MinHash is that we can use random hash functions to perform a coordinated random sample of both sets. Consider choosing the minimum hash value from the union of the sets; that value corresponds to an item in their intersection with probability precisely equal to the Jaccard index, but that is equivalent to whether or not the minimum hash value is the same for both of the two original sets. Thus, overlap in the sets of minimum hash value(s) can be used as an estimator for the Jaccard index. Mash applies MinHash to sets of *k*-mers in genomic or metagenomic sequences as a fast proxy for genomic similarity, and successors such as BinDash[92] and Dashing[93] use other data sketches such as HyperLogLog[94] to similar ends.

Notice that sets of minimum hashes cannot be used to reconstruct the original sets, even approximately, as those sets are sampled from the original sets. With Mash, this transformation is even lossier, because it is only sampling the sets of *k*-mers, and there is no hope of reconstructing the original sequences. Thus, this type of data sketching is only useful for specific downstream analyses, such as set or sequence similarity, and cannot be used as a generic lossy compression — indeed, Mash has proven inaccurate for measuring containment of a genome within metagenomic sequences, for which other tools such as Mash Screen are more effective[95]. However, the flip side is that, very often, we are able to achieve not only massive savings in space (both communication and RAM costs), through for example `Bloom filters`[96], but also significant improvements in analysis speed, for example bitsliced genomic signature index (BIGSI)[97]. Instead of the usual space–time trade-off of compression, we thus get better space-complexity and time-complexity, but instead at the cost of analysis accuracy, especially when storing only a few *k*-mers from very long sequences or if there are many sequences such as with `metagenomics`.

In addition to applying generic set sketching approaches such as MinHash and HyperLogLog to sets of genomic *k*-mers, there are also biological-analysis specific methodologies. Minimizer-based approaches apply something similar to a MinHash sketch along windows of a genome to sample *k*-mer anchors along a sequence for fast secondary analyses, such as mapping, alignment, classification or structural variation detection. Although originally developed in 2003, minimizer-like approaches have only recently become essential, forming the basis of Minimap/Minimap2 (ref.[66]), Kraken 2 (ref.[98]), mashmap2 (ref.[99]), segmental duplication evaluation framework (SEDEF)[100] and minimizer-space de Bruijn graphs (mdBg)[101,102]. A lot of recent work has been done on how to best sample *k*-mers, whether by changing the minimizer hash function[103–105], by using a different *k*-mer selection scheme[106,107] or by preferentially selecting members of a universal hitting set[108,109]. The key is that these are sample methods designed specifically for sequence data, instead of sets. As with set sketching, there is again a trade-off of accuracy, but here the accuracy loss is often barely visible because of the amount of redundancy present in adjacent *k*-mers, which is the primary loss of information.

More recently, sketching has been generalized to biological data besides just sets and sequences. This is essential in single-cell analysis, given the amount of raw data in that domain, and the most relevant trade-off here is between accuracy and speed. Although any sketch will lose information with respect to analysis on the original data, sketches can be tuned so that certain applications get highlighted. For example, the Geosketch[41] and Hopper[110] methods sketch in such a way as to often improve rare cell type detection; by downsampling regions of high cell density in the cell projection plot, the sketch actually improves analysis in regions with sparse data, where rare cell types occur as outliers.

Hardware accelerators

General-purpose computation makes use of computer chips that are designed to be fairly good at running standard algorithms. However, each algorithm makes use of different compute characteristics; some methods may require randomly accessing large data structures, whereas other methods are easily vectorizable and repeatedly perform the same operation to many different inputs. General-purpose computing needs to be reasonably good at all of these tasks, and so tends to not be great at any one.

Many data science applications involve repeatedly performing the same operation on many different inputs; indeed, GPUs and similar hardware accelerators have been instrumental in making deep learning feasible, as that involves applying the same vector operations to large data vectors repeatedly. Many biological applications are also fairly parallelizable (although not to the same degree as deep learning tasks), and over the decades there have been many attempts at translating bioinformatics algorithms to accelerators[10,111–114]. Deep learning is a sufficiently massive undertaking that custom chips in the form of TPUs have been manufactured[115]. However, although chips similar to TPUs have been designed for genomics[114], they have not been manufactured; instead, genomics applications in practice make use of existing commodity GPUs and FPGAs[111–117]. Still, the toolchains surrounding GPU bioinformatics are not as well developed or easy to use. Fully exploiting hardware accelerators thus currently requires not just expertise on the part of the programmers but additional knowledge by end users (or at least the compute infrastructure teams) on how to integrate those accelerators in their pipelines. Given those costs, it is often easier to solve problematic bioinformatics tasks simply by using additional normal cores. In recent years, some new platforms have sought to address the toolchain and user experience problems; for example, Illumina Dragen[116] and Nvidia Clara Parabricks[117] seek to be single integrated solutions for an entire pipeline analysis. These solutions only work for specific popular workflows at the moment and are rather expensive investments for all but the larger sequencing cores, but they illustrate some of the promise of hardware-accelerated bioinformatics.

Cloud parallelization

It is relatively straightforward when running a parallelizable analysis to trade off money and time. Very often, it is possible to split an analysis across two machines and finish twice as fast. However, traditionally, it was necessary to estimate the number of machines needed to complete an analysis within a specified amount of time, and then go out, purchase, set up and maintain all those computers. If that estimate is incorrect, then the compute servers may be overloaded, and this is one of the sources of longer than expected wait times for the end user.

One prominent solution to that problem is the ‘cloud’[54]. Although the cloud is composed of many individual machines, nearly all of which could be purchased by individuals, the key factor is the ease with which additional resources can be provisioned, at least once a user learns a particular cloud platform’s application programming interfaces (APIs)[53]. Instead of purchasing individual machines, individuals rent compute time on cloud servers, paying not only for the cloud provider to maintain the machines but also only renting when those machines are needed. Thus, fast analyses can be done by renting more machines with higher priority for more money. Notably, this enables large-scale re-analysis of existing databases of publicly available data, such as in SERRATUS[8]. Cloud-based biobanks[46–49] can additionally centralize and reduce storage/computation costs: for analyses performed in a shared cloud, only one copy of the underlying data and results needs to exist and be accessible; thus, there is less duplication of resources across institutions. Some prominent examples of these efforts include the US National Cancer Institute (NCI) Genomic Data Commons[118], the US National Human Genome Research Institute (NHGRI) AnVIL[119] and the Common Fund Data Ecosystem[120].

Additionally, the cloud gives end users access to a more diverse array of servers, partially solving the issues of needing specific hardware for methods. For example, some compute servers are designed for high-RAM workloads, whereas others are designed with hardware accelerators such as GPUs. If only a limited number of machines can be purchased by a sequencing centre, they may not be able to exactly allocate the right number of each type of machine; alternatively, they may choose to purchase more generalist machines instead. The advantage of renting time on the cloud is that you can, in theory, always rent exactly the types of machines that best suit your analysis. For example, on Amazon Web Services, DNAnexus and Illumina Dragen, hardware are easily deployed for accelerated sequencing pipeline instances. Alternatively, Cloud Life Sciences and Microsoft Genomics are platforms built on Google Cloud and Microsoft Azure, respectively.

Domain-specific libraries and languages

Easy to use programming frameworks for deep learning models, such as TensorFlow and PyTorch[121,122], have had an immense impact on the wide adoption of deep learning, which corroborates the importance of user-friendly engineering tools. Ultimately, researchers in the field lack good tools for developing and updating software; you typically must choose between a software ecosystem that allows rapid development at the expense of performance and scalability (for example, Python or R) or low-level languages that have higher performance but are harder to develop and maintain (for example, C, C++ or Rust). Existing solutions that try to fill the void between these extremes (for example, MATLAB or Julia) are primarily geared towards numerical computing rather than computational genomics.

As discussed earlier in the ‘Complexity’ section above, one of the challenges pertinent to computational genomics is the degree to which many of the smaller tasks in analysis pipelines sum up to a significant fraction of the computational cost. Part of the reason that deep learning frameworks for accelerators work so well is that there is a clear separation in computational cost between the expensive matrix multiplication of training/inference and the glue code; thus, it is fine for glue to be in a slower language such as Python. The separation is less clear in MATLAB, but engineers very quickly learn to vectorize any time-critical code[123], and Julia was designed from the ground up to make both numerical computations and control flow fast[124].

As such, there have been numerous attempts at simplifying the development of bioinformatics software. Most of these attempts have been in the form of collections of libraries of computational biology primitives for popular languages: these include SeqAn for C++[125], Biopython[126], Rust-Bio[127] and BioJulia[128]. Indeed, we note that Rust, which makes it easier to exploit parallelism, has seen increasing adoption among core bioinformatics software developers as `multicore` machines become ubiquitous[127]. However, although C++, Rust and Julia are extremely performant, relatively few bioinformatics practitioners are comfortable with those languages. On the other hand, Python is an extremely widely adopted language for genomics — for example, prominently for single-cell data in Scanpy[129,130] — but it is too high level to have good performance on glue operations.

The other major direction for simplifying the development of bioinformatics software takes inspiration from projects such as Julia, instead aiming to design domain-specific languages for genomics. One early attempt in this direction was BPipe, a domain-specific language specifically focusing on reproducibly defining the pipeline glue holding other software together[131]. Some of the successors for pipeline glue languages include Snakemake[132], which defines bioinformatics workflows using a variant of Python, or workflow systems[133] such as Galaxy[134], which allow for user-friendly construction of pipelines. These workflow systems have found widespread adoption in bioinformatics. Another future direction comes in the form of more recent domain-specific languages, which try to be one-stop shops for developing entire genomics applications, including both pipeline glue and the expensive big-ticket analyses such as mapping. SARVAVID[135] was designed as a domain-specific language that specifically scaled genomics applications by inherently exploiting parallelism, which in many other languages requires complicated explicit constructs. More recently, the domain-specific language Seq[72] was designed to share the syntax and semantics of Python, making it easy for practitioners to adopt while adding genomics-specific language constructs and having the fast runtime of a compiled language. For the end user, the pipeline glue components are likely to be of greater interest, but modern domain-specific languages hold the promise of reducing the separation between software developers and end users if both adopt the same languages.

Conclusions and future perspectives

In this Review, we have covered typical genomics sequencing pipelines and discussed some of the trade-offs researchers will be called upon to make as the relative cost of compute in these pipelines increases. Technical trade-offs that software developers face include those between computation, storage and communication. User-facing bottlenecks include wait time, monetary cost and the more subtle trade-offs in the complexity of development and deployment of software and infrastructure, and accuracy of the analysis. We have reviewed some of the various new techniques for addressing these trade-offs, focusing on data compression, data sketching, hardware accelerators, cloud parallelization and domain-specific libraries and languages. We also provide user guidance as to how to navigate these trade-offs throughout the Review. We conclude that these modern methodologies are an essential tool in any genomics toolkit as practitioners weigh which trade-offs they are willing to make.

These trade-offs are already a problem for smaller laboratories for whom cloud computing, GPUs and high-RAM machines can be prohibitively expensive — in areas relating to deep learning, there is a well-documented compute divide between those who have access to advanced resources versus those who do not[136,137]. However, we think that the consideration of trade-offs will become especially important for everyone in the single-cell era, because of the raw amount of reads that can be generated from even just a single sample. Furthermore, this decision will be continuous, rather than discrete; many methods allow a continuous trade-off, for example in accuracy versus compute, so researchers will have to decide how much accuracy is ‘enough’, how long they are willing to wait for the computation to complete or how much they are willing to pay a cloud provider.

We look to the large new cloud-based biobanks and single-cell data sources generated by various consortia and cell atlas efforts. As these projects continue, interoperability and standardization questions will be as important as individual computational bottlenecks. Indeed, although some fraction of bottlenecks we discussed come from the ‘hard’ tasks of genomics, some other large fraction of compute and time is spent on just converting from one format to another. The second task of converting formats will become increasingly important as the variety of software and goals grows. Moreover, as with bulk genomic data, it is essential that analytical tools are robust and well maintained, which entails consistent support and improvement to tools beyond what is possible in the conventional single-laboratory academic setting.

These same types of considerations are also being faced by other fields where analytical and computational costs are a significant proportion of total expense. The specific decisions of what trade-offs are beneficial to make will differ, but these kinds of trade-offs are not specific to genomics, or even biology at large.

Acknowledgements

The authors thank E. Banks, L. Cowen and I. Numanagić for helpful discussions. Y.W.Y. is supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) grant RGPIN-2022-03074 and B.B. is supported by US National Institutes of Health (NIH) grant 1R35GM141861.

Glossary

Accelerators

A hardware device or software program that enhances the overall performance of the computer. A software accelerator implements as many system functions as possible in software and moves performance-critical functions into special-purpose external hardware to reduce compute time

Bloom filters

An indexing approach for storing the presence or absence of *k*-mers in a dataset; they have been leveraged to considerably reduce the amount of space and still run in constant time. However, they can have high false positive rates (that is, query hits when there are none)

CIGAR strings

(Concise idiosyncratic gapped alignment report strings). The sequence alignment map (SAM) file format’s compressed representation of a read alignment to a reference

Cloud computing

The use of computing resources distributed in the ‘cloud-shaped’ Internet to store, manage and analyse data, rather than doing so on a local server or personal computer

Complexity

Algorithm complexity is generally measured as an upper bound on its long-term growth rate: how its runtime or space requirements grows as the input size grows, rather than its absolute magnitude, and thus constants are omitted. In practice, a set of algorithms can share the asymptotic complexity despite some of them being a constant 2, 3 or even 1,000 times slower than their counterparts in the set

Compute resources

The amount of compute power (for example, central processing units (CPUs) and memory) that can be requested, allocated and used for computing

Domain-specific languages

Computer languages tailored to a specific domain such as genomics

Field-programmable gate arrays

(FPGAs). Hardware accelerators that can be configured/reprogrammed by a customer after manufacturing. They enable custom hardware acceleration without needing entirely new chips to be manufactured

Graphics processing units

(GPUs). Hardware accelerators that can process many pieces of data simultaneously. They were historically used primarily for rendering computer graphics, but the massive parallelism makes them useful for applications such as machine learning

Jaccard index

A measure of the similarity between two sets, defined as the size of the intersection divided by the size of the union

***k*-mer**

Genomic data normally come in long strings of nucleotides (A, C, G and T). Many genomic algorithms process these strings by looking at exact matches of length-*k* substrings, which are known as *k*-mers

Kryder’s law

Disk drive density doubles every 13 months, determined by the capability of hard drive storage media over time

Lossless compression

A procedure that takes advantage of redundancy/repetition to reversibly transform a large file into a smaller one — for example, storing the string ‘ACGTACGTACGTACGTACGT’ as ‘5*(ACGT)’. Note that although shorter, the transformed string contains all the same information as the original

Lossy compression

Sometimes, we are willing to discard some information when compressing a file. For example, if we start with data points ‘12.362, 15.212, 92.786’ we could round the points and discard some precision to get ‘12, 15, 93’, which can be stored in less space. However, after lossy compression, although we can still reproduce data that look similar to the same kind of format as the original, they are no longer an exact replica

Metagenomics

Ordinary genomics studies the genome of a single organism. Metagenomics is the simultaneous study of a collection of many different species’ genomes in a single sample, typically that of microbial communities

Moore’s law

Computing power (in TeraFLOPS) doubles every 18 months, determined by the number of transistors you can pack per unit area on a chip

Multicore

A single computing processor with two or more independent computing units (called cores). Running multiple instructions on multiple cores at the same time can increase the overall speed of programs

Parallelization

Parallel computing allows numerous calculations to be performed simultaneously, thereby accelerating computation. Based on this principle, many large-scale computational tasks can then be divided into smaller ones and solved on multiple machines concurrently

Parsing

The input data to a computer program can come in various formats. Before performing any type of complicated analysis, programs must first translate those data into an internal representation, in a process known as parsing

Random access memory

(RAM). Short-term storage for data the computer is actively using to speed access

Random access

Access to any element of stored data as easily and efficiently as any other

RNA sequencing

A genomic approach for the detection and quantitative analysis of mRNA molecules in a biological sample

Scale

Scalability typically refers to how an algorithm handles larger amounts of data; for example, an algorithm scales with the amount of data if its runtime and space requirements grow slowly enough in required time and size to solve the problem

Single-threaded

Computation that operates as a single sequential series of operations without any parallelization. It is often used as a benchmark for the speed of a method without using any types of hardware tricks or multi-threaded acceleration

Sketching

These methods reduce the number of data points considered, while still capturing salient features of the underlying data, to minimize the computational resources required for large-scale analyses. Unlike lossy data compression, it is generally not possible to reproduce even an approximate copy of the original data, because the sketch only summarizes a few important features

Space-complexity

Computer scientists traditionally measure the amount of computer memory (random access memory (RAM)) an algorithm needs to run by asking how the amount of memory needed scales with the size of the data. Often, the same types of terms are used as for time-complexity, and we speak of linear, log-linear or quadratic space algorithms

Technology refresh lifecycle

The cycle of regularly updating compute infrastructure to maximize a system's performance

Tensor processing units

(TPUs). Systems developed by Google for application-specific integrated circuits to accelerate machine learning workflows

Time-complexity

Computer scientists traditionally measure how fast an algorithm is by asking how the number of central processing unit (CPU) operations scales with the size of the data. An algorithm is linear time if doubling the amount of data to be processed doubles the number of CPU operations needed. An algorithm is quadratic time if doubling the amount of data quadruples (×4) the number of CPU operations. A log-linear time algorithm is only marginally slower than a linear time algorithm, although the exact scaling requires a bit more mathematical formalism to describe. Most practical algorithms are either linear or log-linear

References

1. Wetterstrand KA DNA sequencing costs: data. National Human Genome Research Institute www.genome.gov/sequencingcostsdata (2022).
2. Preston J, VanZeeland A, & Peiffer DA Innovation at illumina: the road to the $600 human genome. Nature Portfolio https://www.nature.com/articles/d42473-021-00030-9 (2021).
3. Pennisi EA $100 genome? New DNA sequencers could be a 'game changer' for biology, medicine. Science 376, 1257–1258 (2022). [PubMed: 35709273]
4. Regalado A China's BGI says it can sequence a genome for just $100. MIT Technology Review. https://www.technologyreview.com/2020/02/26/905658/china-bgi-100-dollargenome/ (2020).
5. Berger B, Daniels NM & Yu YW Computational biology in the 21st century: scaling with compressive algorithms. Commun. ACM 59, 72–80 (2016). [PubMed: 28966343]
6. Rozenblatt-Rosen O, Stubbington MJT, Regev A & Teichmann SA The Human Cell Atlas: from vision to reality. Nature 550, 451–453 (2017). [PubMed: 29072289]

7. Zheng G Our 1.3 million single cell dataset is ready to download. 10x Genomics. https://www.10xgenomics.com/blog/our-13-million-single-cell-dataset-is-ready-to-download (2022).
8. Edgar RC et al. Petabase-scale sequence alignment catalyses viral discovery. Nature 602, 142–147 (2022). [PubMed: 35082445]
9. Marçais G, Solomon B, Patro R & Kingsford C Sketching and sublinear data structures in genomics. Annu. Rev. Biomed. Data Sci 2, 93–118 (2019).This work is an excellent in-depth review of sketching for algorithm designers.
10. Kurzak J, Bader DA, & Dongarra J, (eds) Scientific Computing with Multicore and Accelerators (CRC, 2010 Dec 7).
11. Mernik M, Heering J & Sloane AM When and how to develop domain-specific languages. ACM Comput. Surv 37, 316–344 (2005).
12. Van der Auwera GA et al. From FastQ data to high-confidence variant calls: the genome analysis toolkit best practices pipeline. Curr. Protoc. Bioinforma 43, 11 (2013).
13. McKenna A et al. The genome analysis toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. Genome Res 20, 1297–1303 (2010). [PubMed: 20644199]
14. Banks E Run the germline GATK best practices pipeline for $5 per genome. GitHub https://github.com/broadinstitute/gatk-docs/blob/master/blog-2012-to-2019/2018-02-12-Run_the_germline_GATK_Best_Practices_Pipeline_for_%245_per_genome.md (2020).
15. Illumina. DRAGEN Complete Suite; latest version: 4.0.3 AWS Marketplace. https://aws.amazon.com/marketplace/pp/prodview-ypz2tpzy6f5xq (2022).
16. Shajii A, Yorukoglu D, Yu YW & Berger B Fast genotyping of known SNPs through approximate *k*-mer matching. Bioinformatics 32, i538–i544 (2016). [PubMed: 27587672]
17. Ondov BD et al. Mash: fast genome and metagenome distance estimation using MinHash. Genome Biol 17, 1–4 (2016). [PubMed: 26753840]
18. Stein L Genome annotation: from sequence to biology. Nat. Rev. Genet 2, 493–503 (2001). [PubMed: 11433356]
19. Lewis CM Genetic association studies: design, analysis and interpretation. Brief. Bioinforma 3, 146–153 (2002).
20. Baldi P & Brunak S Bioinformatics: The Machine Learning Approach (MIT Press, 2001).
21. Suhre K, McCarthy MI & Schwenk JM Genetics meets proteomics: perspectives for large population-based studies. Nat. Rev. Genet 22, 19–37 (2021). [PubMed: 32860016]
22. Allis DC & Jenuwein T The molecular hallmarks of epigenetic control. Nat. Rev. Genet 17, 487–500 (2016). [PubMed: 27346641]
23. Moses L & Pachter L Museum of spatial transcriptomics. Nat. Methods 19, 534–546 (2022). [PubMed: 35273392]
24. Burgess DJ Spatial transcriptomics coming of age. Nat. Rev. Genet 20, 317–317 (2019). [PubMed: 30980030]
25. Berger B & Cho H Emerging technologies towards enhancing privacy in genomic data sharing. Genome Biol 20, 1–3 (2019). [PubMed: 30606230]
26. Gürsoy G et al. Functional genomics data: privacy risk assessment and technological mitigation. Nat. Rev. Genet 2021, 1–14 (2021).
27. Cormen TH, Leiserson CE, Rivest RL, & Stein C Introduction to Algorithms (MIT Press, 2022).
28. Tomczak K, Czerwińska P & Wiznerowicz M The Cancer Genome Atlas (TCGA): an immeasurable source of knowledge. Contemp. Oncol 19, A68–A77 (2015).
29. Zhang Z et al. Uniform genomic data analysis in the NCI Genomic Data Commons. Nat. Commun 12, 1226 (2021). [PubMed: 33619257]
30. BackupWorks.com. LTO Program announces price per gigabyte now less than one penny BackupWorks.com https://www.backupworks.com/LTO-programcost-per-gigabyte-milestone.aspx (2022).
31. 100,000 Genomes Project Pilot Investigators. 100,000 genomes pilot on rare-disease diagnosis in health care — preliminary report. N. Engl. J. Med 385, 1868–1880 (2021). [PubMed: 34758253]
32. Matange K, Tuck JM & Keung AJ DNA stability: a central design consideration for DNA data storage systems. Nat. Commun 12, 1358 (2021). [PubMed: 33649304]

33. Jacob B, Wang D, & Ng S Memory Systems: Cache, DRAM, disk (Morgan Kaufmann, 2010).
34. Bonfield JK CRAM 3.1: advances in the CRAM file format. Bioinformatics 38, 1497–1503 (2022). [PubMed: 34999766]
35. Li H et al. The sequence alignment/map format and SAMtools. Bioinformatics 25, 2078–2079 (2009). [PubMed: 19505943]
36. Cock PJ, Fields CJ, Goto N, Heuer ML & Rice PM The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. Nucleic Acids Res 38, 1767–1771 (2010). [PubMed: 20015970]
37. Hernaez M, Pavlichin D, Weissman T & Ochoa I Genomic data compression. Annu. Rev. Biomed. Data Sci 2, 19–37 (2019).This work is a canonical review of genomic data compression by many of the authors involved in standardization efforts.
38. Loh PR, Baym M & Berger B Compressive genomics. Nat. Biotechnol 30, 627–630 (2012). [PubMed: 22781691]
39. Langmead B & Nellore A Cloud computing for genomic data analysis and collaboration. Nat. Rev. Genet 19, 208–219 (2018). [PubMed: 29379135] This article goes more in-depth on cloud computing and how that is changing genomic data analysis.
40. Danecek P et al. The variant call format and VCFtools. Bioinformatics 27, 2156–2158 (2011). [PubMed: 21653522]
41. Hie B, Cho H, DeMeo B, Bryson B & Berger B Geometric sketching compactly summarizes the single-cell transcriptomic landscape. Cell Syst 8, 483–493 (2019). [PubMed: 31176620]
42. Hie B et al. Computational methods for single-cell RNA sequencing. Annu. Rev. Biomed. Data Sci 3, 339–364 (2020).This review discusses some of the newer computational challenges presented by scRNA-seq data.
43. Lähnemann D et al. Eleven grand challenges in single-cell data science. Genome Biol 21, 1–35 (2020).
44. Evans C, Hardin J & Stoebel DM Selecting between-sample RNA-seq normalization methods from the perspective of their assumptions. Brief. Bioinforma 19, 776–792 (2018).
45. Google. All networking pricing Google Cloud https://cloud.google.com/vpc/networkpricing (2022).
46. Bycroft C et al. The UK Biobank resource with deep phenotyping and genomic data. Nature 562, 203 (2018). [PubMed: 30305743]
47. Chen Z et al. China Kadoorie Biobank of 0.5 million people: survey methods, baseline characteristics and long-term follow-up. Int. J. Epidemiol 40, 1652–1666 (2011). [PubMed: 22158673]
48. Gaziano JM et al. Million veteran program: a mega-biobank to study genetic influences on health and disease. J. Clin. Epidemiol 70, 214–223 (2016). [PubMed: 26441289]
49. Lin JC, Hsiao WWW & Fan CT Transformation of the Taiwan Biobank 3.0: vertical and horizontal integration. J. Transl. Med 18, 1–13 (2020). [PubMed: 31900168]
50. All of Us Research Program Investigators. The "All of Us" research program. N. Engl. J. Med 381, 668–676 (2019). [PubMed: 31412182]
51. Baker M & Buyya R Cluster computing: the commodity supercomputer. Softw. Pract. Exp 29, 551–576 (1999).
52. Goenka SD et al. Accelerated identification of disease-causing variants with ultra-rapid nanopore genome sequencing. Nat. Biotechnol 40, 1035–1041 (2022). [PubMed: 35347328]
53. Marshall P, Keahey K, & Freeman T in 2011 11th IEEE/ACM Int. Symp. Cluster, Cloud and Grid Computing 205–214 (IEEE, 2011).
54. Grossman RL The case for cloud computing. IT professional 11, 23–27 (2009).
55. Cormode G & Garofalakis M in Proc. 2007 ACM SIGMOD Int. Conf. Management of Data 281–292 (2007).
56. Altschul SF, Gish W, Miller W, Myers EW & Lipman DJ Basic local alignment search tool. J. Mol. Biol 215, 403–410 (1990). [PubMed: 2231712]
57. Smith TF & Waterman MS Identification of common molecular subsequences. J. Mol. Biol 147, 195–197 (1981). [PubMed: 7265238]

58. Berger B, Waterman MS & Yu YW Levenshtein distance, sequence comparison and biological database search. IEEE Trans. Inf. Theory 67, 3287–3294 (2020). [PubMed: 34257466]

59. He D et al. Alevin-fry unlocks rapid, accurate and memory-frugal quantification of single-cell RNA-seq data. Nat. Methods 19, 316–322 (2022). [PubMed: 35277707]

60. Kaminow B, Yunusov D & Dobin A STARsolo: accurate, fast and versatile mapping/quantification of single-cell and single-nucleus RNA-seq data. Preprint at Biorxiv 10.1101/2021.05.05.442755 (2021).

61. Sarkar H, Srivastava A & Patro R Minnow: a principled framework for rapid simulation of dscRNA-seq data at the read level. Bioinformatics 35, i136–i144 (2019). [PubMed: 31510649]

62. Regier AA et al. Functional equivalence of genome sequencing analysis pipelines enables harmonized variant calling across human genetics projects. Nat. Commun 9, 1–8 (2018). [PubMed: 29317637]

63. Kent WJ BLAT — the BLAST-like alignment tool. Genome Res 12, 656–664 (2002). [PubMed: 11932250]

64. Li H Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. Preprint at arXiv 10.48550/arXiv.1303.3997 (2013).

65. Langmead B & Salzberg SL Fast gapped-read alignment with Bowtie 2. Nat. Methods 9, 357–359 (2012). [PubMed: 22388286]

66. Li H Minimap2: pairwise alignment for nucleotide sequences. Bioinformatics 34, 3094–3100 (2018). [PubMed: 29750242]

67. Grigoryev DN in Big Data Analysis for Bioinformatics and Biomedical Discoveries (ed. Ye SQ) 15–34 (CRC, 2016).

68. Korsunsky I et al. Fast, sensitive and accurate integration of single-cell data with Harmony. Nat. Methods 16, 1289–1296 (2019). [PubMed: 31740819]

69. Stuart T et al. Comprehensive integration of single-cell data. Cell 177, 1888–1902 (2019). [PubMed: 31178118]

70. Endrullat C, Glökler J, Franke P & Frohme M Standardization and quality management in next-generation sequencing. Appl. Transl. Genomics 10, 2–9 (2016).

71. Yorukoglu D, Yu YW, Peng J & Berger B Compressive mapping for next-generation sequencing. Nat. Biotechnol 34, 374–376 (2016). [PubMed: 27054987]

72. Shajii A et al. A Python-based programming language for high-performance computational genomics. Nat. Biotechnol 39, 1062–1064 (2021). [PubMed: 34282326]

73. Berger B, Peng J & Singh M Computational solutions for omics data. Nat. Rev. Genet 14, 333–346 (2013). [PubMed: 23594911] This work is an older review of computational challenges and solutions in bioinformatics, the topics of which this Review assumes background familiarity with.

74. Rehm HL et al. GA4GH: international policies and standards for data sharing across genomic research and healthcare. Cell Genomics 1, 100029 (2021). [PubMed: 35072136]

75. Alberti C et al. in Proc. IEEE Data Compression Conf. (DCC) 221–230 (2016).

76. Fritz MH, Leinonen R, Cochrane G & Birney E Efficient storage of high throughput DNA sequencing data using reference-based compression. Genome Res 21, 734–740 (2011). [PubMed: 21245279]

77. Bonfield JK & Mahoney MV Compression of FASTQ and SAM format sequencing data. PloS ONE 8, e59190 (2013). [PubMed: 23533605]

78. Rahman A, Chikhi R & Medvedev P Disk compression of *k*-mer sets. Algorithms Mol. Biol 16, 1–4 (2021). [PubMed: 33639968]

79. Hach F, Numanagić I, Alkan C & Sahinalp SC SCALCE: boosting sequence compression algorithms using locally consistent encoding. Bioinformatics 28, 3051–3057 (2012). [PubMed: 23047557]

80. Janin L, Schulz-Trieglaff O & Cox AJ BEETL-fastq: a searchable compressed archive for DNA reads. Bioinformatics 30, 2796–2801 (2014). [PubMed: 24950811]

81. Yu YW, Daniels NM, Danko DC & Berger B Entropy-scaling search of massive biological data. Cell Syst 1, 130–140 (2015). [PubMed: 26436140]

82. Ferragina P & Manzini G in Proc. 41st Annual Symp. Foundations of Computer Science 390–398 (IEEE, 2000).

83. Ferragina P, Manzini G, Mäkinen V & Navarro G Compressed representations of sequences and full-text indexes. ACM Trans. Algorithms 10.1145/1240233.1240243 (2007).

84. Kuhnle A et al. Efficient construction of a complete index for pan-genomics read alignment. J. Comput. Biol 27, 500–513 (2020). [PubMed: 32181684]

85. Bhaskaran V & Konstantinides K Image and Video Compression Standards: Algorithms and Architectures (Springer, 1997).

86. Yu YW, Yorukoglu D, Peng J & Berger B Quality score compression improves genotyping accuracy. Nat. Biotechnol 33, 240–243 (2015). [PubMed: 25748910]

87. Malysa G et al. QVZ: lossy compression of quality values. Bioinformatics 31, 3122–3129 (2015). [PubMed: 26026138]

88. Ochoa I, Hernaez M, Goldfeder R, Weissman T & Ashley E Effect of lossy compression of quality scores on variant calling. Brief. Bioinforma 18, 183–194 (2017).

89. Broder AZ in IEEE Proc. Compression and Complexity of SEQUENCES (Cat. No.97TB100171) 21–29 (IEEE, 1997).

90. Broder AZ, Charikar M, Frieze AM & Mitzenmacher M in Proc. 30th ACM Symp. Theory of Computing (STOC '98) 327–336 (Association for Computing Machinery, 1998).

91. Jaccard P The distribution of the flora in the alpine zone. N. Phytol 11, 37–50 (1912).

92. Zhao X BinDash, software for fast genome distance estimation on a typical personal laptop. Bioinformatics 35, 671–673 (2019). [PubMed: 30052763]

93. Baker DN & Langmead B Dashing: fast and accurate genomic distances with HyperLogLog. Genome Biol 20, 265 (2019). [PubMed: 31801633]

94. Flajolet P, Fusy É, Gandouet O & Meunier F Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. Discret. Math. Theor. Comput. Sci 10.46298/dmtcs.3545 (2007).

95. Ondov BD et al. Mash Screen: high-throughput sequence containment estimation for genome discovery. Genome Biol 20, 1–3 (2019). [PubMed: 30606230]

96. Stranneheim H et al. Classification of DNA sequences using Bloom filters. Bioinformatics 26, 1595–1600 (2010). [PubMed: 20472541]

97. Bradley P et al. Ultrafast search of all deposited bacterial and viral genomic data. Nat. Biotechnol 37, 152–159 (2019). [PubMed: 30718882]

98. Wood DE, Lu J & Langmead B Improved metagenomic analysis with Kraken 2. Genome Biol 20, 1–3 (2019). [PubMed: 30606230]

99. Jain C, Koren S, Dilthey A, Phillippy AM & Aluru S A fast adaptive algorithm for computing whole-genome homology maps. Bioinformatics 34, i748–i756 (2018). [PubMed: 30423094]

100. Numanagić I et al. Fast characterization of segmental duplications in genome assemblies. Bioinformatics 34, i706–i714 (2018). [PubMed: 30423092]

101. Ekim B, Berger B & Chikhi R Minimizer-space de Bruijn graphs: whole-genome assembly of long reads in minutes on a personal computer. Cell Syst 12, 958–968 (2021). [PubMed: 34525345]

102. Rautiainen M & Marschall T MBG: minimizer-based sparse de Bruijn Graph construction. Bioinformatics 37, 2476–2478 (2021). [PubMed: 33475133]

103. Marçais G et al. Improving the performance of minimizers and winnowing schemes. Bioinformatics 33, i110–i117 (2017). [PubMed: 28881970]

104. Jain C et al. Weighted minimizer sampling improves long read mapping. Bioinformatics 36, i111–i118 (2020). [PubMed: 32657365]

105. Flomin D, Pellow D & Shamir R Data set-adaptive minimizer order reduces memory usage in *k*-mer counting. J. Comput. Biol 29, 825–838 (2022). [PubMed: 35527644]

106. Edgar R Syncmers are more sensitive than minimizers for selecting conserved k-mers in biological sequences. PeerJ 9, e10805 (2021). [PubMed: 33604186]

107. Shaw J & Yu YW Theory of local *k*-mer selection with applications to long-read alignment. Bioinformatics 2021, btab790 (2021).

108. Orenstein Y, Pellow D, Marçais G, Shamir R & Kingsford C Designing small universal k-mer hitting sets for improved analysis of high-throughput sequencing. PLoS Comput. Biol 13, e1005777 (2017). [PubMed: 28968408]

109. Ekim B, Berger B & Orenstein Y in Proc. Int. Conf. Research in Computational Molecular Biology (RECOMB) (ed. Schwartz R) 37–53 (Springer LNBI, 2020).

110. DeMeo B & Berger B Hopper: a mathematically optimal algorithm for sketching biological data. Bioinformatics 36, i236–i241 (2020). [PubMed: 32657375]

111. Manavski SA & Valle G CUDA compatible GPU cards as efficient hardware accelerators for Smith–Waterman sequence alignment. BMC Bioinforma 9, 1–9 (2008).

112. Herbordt MC, Model J, Gu Y, Sukhwani B & VanCourt T in Proc. 14th Annual IEEE Symp. Field-Programmable Custom Computing Machines Vol. 2006 217–226 (IEEE, 2006).

113. Alser M, Shahroodi T, Gómez-Luna J, Alkan C & Mutlu O SneakySnake: a fast and accurate universal genome pre-alignment filter for CPUs, GPUs and FPGAs. Bioinformatics 36, 5282–5290 (2020).

114. Cali DS et al. in 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO) 951–966 (IEEE, 2020).

115. Jouppi NP et al. in Proc. 44th Annual Int. Symp. Computer Architecture Vol. 24 1–12 (2017).

116. Catreux S et al. DRAGEN Sets New Standard for Data Accuracy in Precision FDA Benchmark Data. Optimizing Variant Calling Performance with Illumina Machine Learning and DRAGEN Graph Illumina https://www.illumina.com/science/genomics-research/articles/dragen-shines-again-precisionfda-truth-challenge-v2.html (2020).

117. NVIDIA. Genome sequencing analysis NVIDIA https://www.nvidia.com/en-us/clara/genomics/ (2022).

118. Heath AP et al. The NCI Genomic Data Commons. Nat. Genet 53, 257–262 (2021). [PubMed: 33619384]

119. Schatz MC et al. Inverting the model of genomics data sharing with the NHGRI genomic data science analysis, visualization, and informatics lab-space. Cell Genomics 2, 100085 (2022). [PubMed: 35199087]

120. Charbonneau AL et al. Making Common Fund data more findable: catalyzing a data ecosystem. Preprint at bioRxiv 10.1101/2021.11.05.467504 (2021).

121. Abadi M et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. Preprint at arXiv https://arxiv.org/abs/1603.04467 (2016).

122. Paszke A et al. Pytorch: an imperative style, high-performance deep learning library. Adv. Neural Inf. Process. Syst 32, 8026–8037 (2019).

123. Gjendemsjø A An introduction to MATLAB OpenStax CNX http://cnx.org/contents/2100a51e-a5c9-4e41-9cb6-087b755125ac@3.4 (2007).

124. Perkel JM Julia: come for the syntax, stay for the speed. Nature 572, 141–143 (2019). [PubMed: 31363196]

125. Döring A et al. SeqAn an efficient, generic C++ library for sequence analysis. BMC Bioinforma 9, 11 (2008).

126. Cock PJA et al. Biopython: freely available Python tools for computational molecular biology and bioinformatics. Bioinformatics 25, 1422–1423 (2009). [PubMed: 19304878]

127. Köster J Rust-Bio: a fast and safe bioinformatics library. Bioinformatics 32, 444–446 (2016). [PubMed: 26446134]

128. Ward BJ Fast, open, easy, software for biology BioJulia https://biojulia.net (2022).

129. Angerer P et al. Single cells make big data: new challenges and opportunities in transcriptomics. Curr. Opin. Syst. Biol 4, 85–91 (2017).

130. Wolf F, Angerer P & Theis F SCANPY: large-scale single-cell gene expression data analysis. Genome Biol 19, 15 (2018). [PubMed: 29409532]

131. Saledin SP, Pope B & Oshlack A BPipe: a tool for running and managing bioinformatics pipelines. Bioinformatics 28, 1525–1526 (2012). [PubMed: 22500002]

132. Köster J & Rahmann S Snakemake — a scalable bioinformatics workflow engine. Bioinformatics 28, 2520–2522 (2012). [PubMed: 22908215]

133. Reiter T et al. Streamlining data-intensive biology with workflow systems. GigaScience 10, giaa140 (2021).

134. Blankenberg D et al. Galaxy: a web‐based genome analysis tool for experimentalists. Curr. Protoc. Mol. Biol 89, 19 (2010).

135. Mahadik K et al. Sarvavid: a domain specific language for developing scalable computational genomics applications. Proc. 2016 Int. Conf. Supercomput 10.1145/2925426.2926283 (2016).

136. Ahmed N & Wahed M The de-democratization of AI: deep learning and the compute divide in artificial intelligence research. Preprint at arXiv https://arxiv.org/abs/2010.15581 (2020).

137. Hellendoorn VJ & Sawant AA The growing cost of deep learning for source code. Commun. ACM 65, 31–33 (2021).

138. Shendure J & Ji H Next-generation DNA sequencing. Nat. Biotechnol 26, 1135–1145 (2008). [PubMed: 18846087]

139. Pfeiffer F et al. Systematic evaluation of error rates and causes in short samples in next-generation sequencing. Sci. Rep 8, 1–4 (2018). [PubMed: 29311619]

140. Lang D et al. Comparison of the two up-to-date sequencing technologies for genome assembly: HiFi reads of Pacific Biosciences Sequel II system and ultralong reads of Oxford Nanopore. GigaScience 9, giaa123 (2020).

141. Wenger AM et al. Accurate circular consensus long-read sequencing improves variant detection and assembly of a human genome. Nat. Biotechnol 37, 1155–1162 (2019). [PubMed: 31406327]

142. Workman RE et al. Nanopore native RNA sequencing of a human poly(A) transcriptome. Nat. Methods 16, 1297–1305 (2019). [PubMed: 31740818]

143. Oxford Nanopore. Oxford Nanopore Tech update: new Duplex method for Q30 nanopore single molecule reads, PromethION 2, and more. Oxford Nanopore Technologies https://nanoporetech.com/about-us/news/oxford-nanopore-tech-update-new-duplexmethod-q30-nanopore-single-molecule-reads-0 (2021).

144. Zheng G et al. Haplotyping germline and cancer genomes with high-throughput linked-read sequencing. Nat. Biotechnol 34, 303–311 (2016). [PubMed: 26829319]

145. Belton JM et al. Hi-C: a comprehensive technique to capture the conformation of genomes. Methods 58, 268–276 (2012). [PubMed: 22652625]

146. Solomon B & Kingsford C Fast search of thousands of short-read sequencing experiments. Nat. Biotechnol 34, 300–302 (2016). [PubMed: 26854477]

147. Sahlin K & Medvedev P De novo clustering of long-read transcriptome data using a greedy, quality value-based algorithm. J. Comput. Biol 27, 472–484 (2020). [PubMed: 32181688]

148. Mohamed S & Syed BA Commercial prospects for genomic sequencing technologies. Nat. Rev. Drug Disco 12, 341 (2013).

149. Eisenstein M Illumina swallows PacBio in long shot for market domination. Nat. Biotechnol 37, 3–5 (2019). [PubMed: 30605147]

150. Sundquist A, Ronaghi M, Tang H, Pevzner P & Batzoglou S Whole-genome sequencing and assembly with high-throughput, short-read technologies. PloS ONE 2, e484 (2007). [PubMed: 17534434]

151. Van Dijk EL, Jaszczyszyn Y, Naquin D & Thermes C The third revolution in sequencing technology. Trends Genet 34, 666–681 (2018). [PubMed: 29941292]

152. Tan G et al. Long fragments achieve lower base quality in Illumina paired-end sequencing. Sci. Rep 9, 2856 (2019). [PubMed: 30814542]

153. Schirmer M et al. Illumina error profiles: resolving fine-scale variation in metagenomic sequencing data. BMC Bioinforma 17, 125 (2016).

154. Dohm JC, Peters P, Stralis-Pavese N & Himmelbauer H Benchmarking of long-read correction methods. NAR Genomics Bioinforma 2, Iqaa037 (2020).

155. Fullwood MJ, Wei CL, Liu ET & Ruan Y Next-generation DNA sequencing of paired-end tags (PET) for transcriptome and genome analyses. Genome Res 19, 521–532 (2009). [PubMed: 19339662]

156. Duan Z et al. A three-dimensional model of the yeast genome. Nature 465, 363–367 (2010). [PubMed: 20436457]

157. Burton JN et al. Chromosome-scale scaffolding of de novo genome assemblies based on chromatin interactions. Nat. Biotechnol 31, 1119–1125 (2013). [PubMed: 24185095]

158. Spies N et al. Genome-wide reconstruction of complex structural variants using read clouds. Nat. Methods 14, 915–920 (2017). [PubMed: 28714986]

159. Loose M, Malla S & Stout M Real-time selective sequencing using nanopore technology. Nat. Methods 13, 751–754 (2016). [PubMed: 27454285]

Related links

BEETL: https://github.com/BEETL/BEETL

BEETL-fastq: https://github.com/BEETL/BEETL

BIGSI: https://github.com/phelimb/BIGSI

BinDash: https://github.com/zhaoxiaofei/BinDash

BPipe: https://github.com/ssadedin/bpipe

CORA: http://cb.csail.mit.edu/cb/cora/

Dashing: https://github.com/dnbaker/dashing

DNAnexus: https://www.dnanexus.com/

DSRC2: http://sun.aei.polsl.pl/dsrc

ESS-Compress: http://github.com/medvedevgroup/ESSCompress

FACS: https://github.com/SciLifeLab/facs

Geosketch: https://geosketch.csail.mit.edu/

Hopper: http://hopper.csail.mit.edu/

Illumina Dragen: https://www.illumina.com/products/by-type/informatics-products/dragenbio-it-platform.html

Illumina Dragen AMI: https://aws.amazon.com/quickstart/architecture/illumina-dragen/

IsONclust: https://pypi.org/project/isONclust/

Kraken 2: https://github.com/DerrickWood/kraken2

Mash: https://github.com/marbl/mash

MashMap2: https://github.com/marbl/MashMap

MBG: https://github.com/maickrau/MBG

mdbg: https://github.com/ekimb/rust-mdbg/

Minimap2: https://github.com/lh3/minimap2

Nvidia Clara Parabricks: https://www.nvidia.com/en-us/clara/genomics/

Qualcomp: https://sourceforge.net/projects/qualcomp

Quartz: https://quartz.csail.mit.edu/

QVZ: https://github.com/mikelhernaez/qvz

SCALCE: http://scalce.sourceforge.net/

SEDEF: https://github.com/vpc-ccg/sedef

Seq: https://github.com/seq-lang/seq

Sequence Bloom Tree: http://www.cs.cmu.edu/~ckingsf/software/bloomtree/

SneakySnake: https://github.com/CMU-SAFARI/SneakySnake

Winnowmap: https://github.com/marbl/Winnowmap

Box 1

Genomic data acquisition: primary data generation

There are a wide variety of sequencing technologies in use, which not only produce large quantities of data but also have considerably different properties (Table 1). Harnessing their potential requires an understanding of their differences and substantial changes in algorithm design.

Short reads: second-generation sequencing

Second-generation sequencing, which comprises the bulk of sequencing data available today, is in the form of short reads of length 200–400 bp. Illumina is the largest commercial provider of short-read sequencing technologies[148,149]. Although second-generation sequencers generate much shorter reads than earlier first-generation (Sanger) technologies, they are also orders of magnitude less expensive[150].

However, the largest human chromosome is about 250 million bp long, and even simple structural variants, such as transposons, are many thousands of base pairs in length. The metadata associated with short reads, especially in the form of ‘paired ends’ where two reads are linked together spatially (because they are two ends of a single physical DNA fragment of length ~1,000 bp), can help with analysis, but many algorithms for analysing genomes have thus had to be customized for interpreting many short overlapping reads.

Long reads: third-generation sequencing

Recent single-molecule sequencing technologies produce long reads (for example, Pacific Biosciences (PacBio) high fidelity (HiFi) and Oxford Nanopore Technologies (ONT)), which are sometimes long enough to span repetitive parts of a genome, thus addressing one of the main limitations of second-generation technologies[151]. Many algorithms and code written for second-generation sequencing assume short reads of 200–400 bp, whereas they may not scale well for long-read technologies, which are able to access lengths greater than 10,000 bp (10 kbp).

Furthermore, each technology has its own distinct error profile. Illumina second-generation sequencers have ~0.24% error[152], primarily in the form of substitutions, rather than insertions and deletions (indels)[153]. On the other hand, raw reads from ONT and PacBio continuous long read (CLR) platforms both had error rates greater than 5%, but PacBio has a much higher rate of insertions whereas ONT errors are more evenly distributed between substitutions, insertions and deletions[154]. These error profiles and rates are also still evolving as companies perfect their methodologies: newer PacBio HiFi circular consensus sequencing and ONT Minion Duplex can reduce error rates to <1%[141,143].

Linking distance reads

Other modern techniques can produce additional information linking together reads[155]. Even early on, Illumina short reads came in tagged pairs, spaced about 1,000 bp apart. Alternatively, Illumina’s Hi-C technology was initially developed to interrogate the 3D structure of chromosomes[145,156]. It generates data linking together pairs of much more distant reads but does not provide information about the distance between them or their relative orientation. The inference of distances between reads along the genome requires algorithms that analyse the read distributions, and specifically the density of pairs linking together separate regions[157]. More recently, the 'read cloud' or 'linked read' approach[158], developed by 10x Genomics (now discontinued) and BGI's Complete Genomics, generates collections of sequencing reads that all originate from the same region of the genome, but without providing any additional information linking specific reads to each other or to specific locations along a chromosome. This additional data can be extremely helpful in the de novo reconstruction of nearly complete genomes.

Real-time data generation

Traditionally, secondary processing of the reads takes place after the sequencing is finished. However, this is no longer always the case. Nanopore-based sequencing devices are capable of generating data in real time, producing a signal as individual DNA molecules pass through a pore in a membrane; thus, computation is now a major bottleneck. Even the algorithms that are used to convert the electrical signal into a read-out of the molecule as a string of nucleotides can barely keep up with the data generation[159], let alone the more complex algorithms needed to use the resulting data to address biological questions. For example, many analyses require the sequences to be compared with other sequences in large reference databases, yet the algorithms used for real-time analyses, such as those used in ONT's Read Until adaptive sampling, are not guaranteed to produce the same output. This is thus a very real user-facing trade-off, in that if they want real-time results, the user cannot use the most accurate algorithms.

Box 2

Genomic data analysis: secondary processing steps

Reads from a sequencer

The sequencer outputs a set of reads. Coloured dots represent distinctive substrings that can be used to work out the locations of the reads in the genome. For display purposes, we simplified the picture and only show up to two dots per read. Additionally, mutations and sequencing errors may cause these substrings to not be present, so differences in (or lack of) coloured dots on a sequence can be interpreted as genetic variants (see the figure, part **a**).

De novo assembly

De novo assembly is the task of reconstructing the original genome from reads without any kind of guiding reference. Relative to reference-based methods, de novo assembly is less prone to reference biases, but is also more expensive. Genome assembly methods are experiencing a resurgence due to long-read technologies increasing overlap relative to short-read sequencing. The output is a graph because bubbles form when two reads that assemble to the same position differ, depicted in the assembly graph figure as a read lacking a coloured dot (see the figure, part **b**).

Read mapping

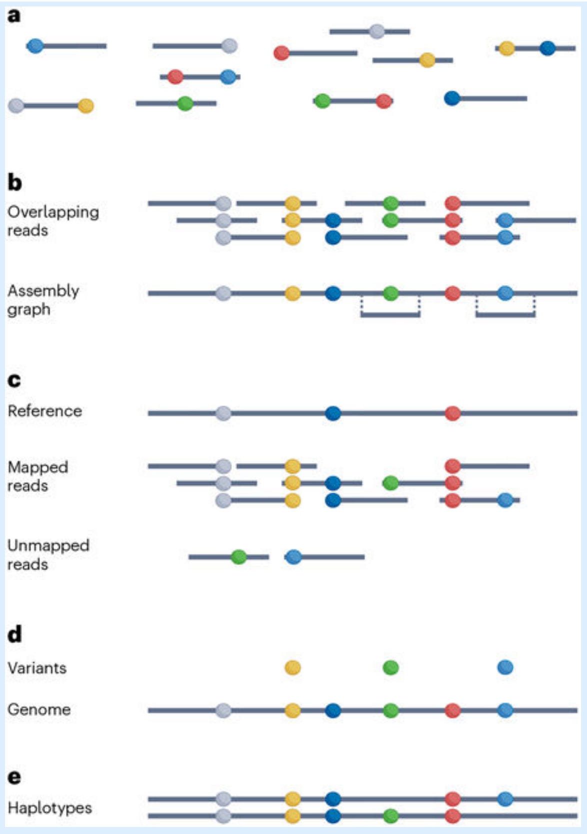
Read mapping locally aligns each read to a similar location; it is generally much faster than assembly and can be easily parallelized as each read is mapped separately. In most sequencing pipelines, mapping reads to a reference genome is the first step performed. In the figure, the coloured dots are used to align reads to matching positions on the genome (see the figure, part **c**).

Variant calling

This step determines the genetic variants present in a sample. Traditional methods rely on finding differences between mapped reads and the reference, which are good for small mutations but not appropriate for large structural changes. Alternatives include assembly-based methods, and *k*-mer substring-based methods. We use the mapped locations from the figure, part **c** to find the coloured dots not present on the reference, the latter of which are the variants present in the sequenced genome; thus, the sequenced genome is the reference from the figure, part **c** plus the additional dots for the variants (see the figure, part **d**).

Phasing/haplotyping

Many species have similar copies of chromosomes, which presents additional challenges for methods based around finding similar locations in a genome. Phasing is the task of disambiguating the multiple copies (or haplotypes), for example, to identify which allele (coloured dot) came from which haplotype (see the figure, part **e**).


a
b
Overlapping reads
Assembly graph
c
Reference
Mapped reads
Unmapped reads
d
Variants
Genome
e
Haplotypes

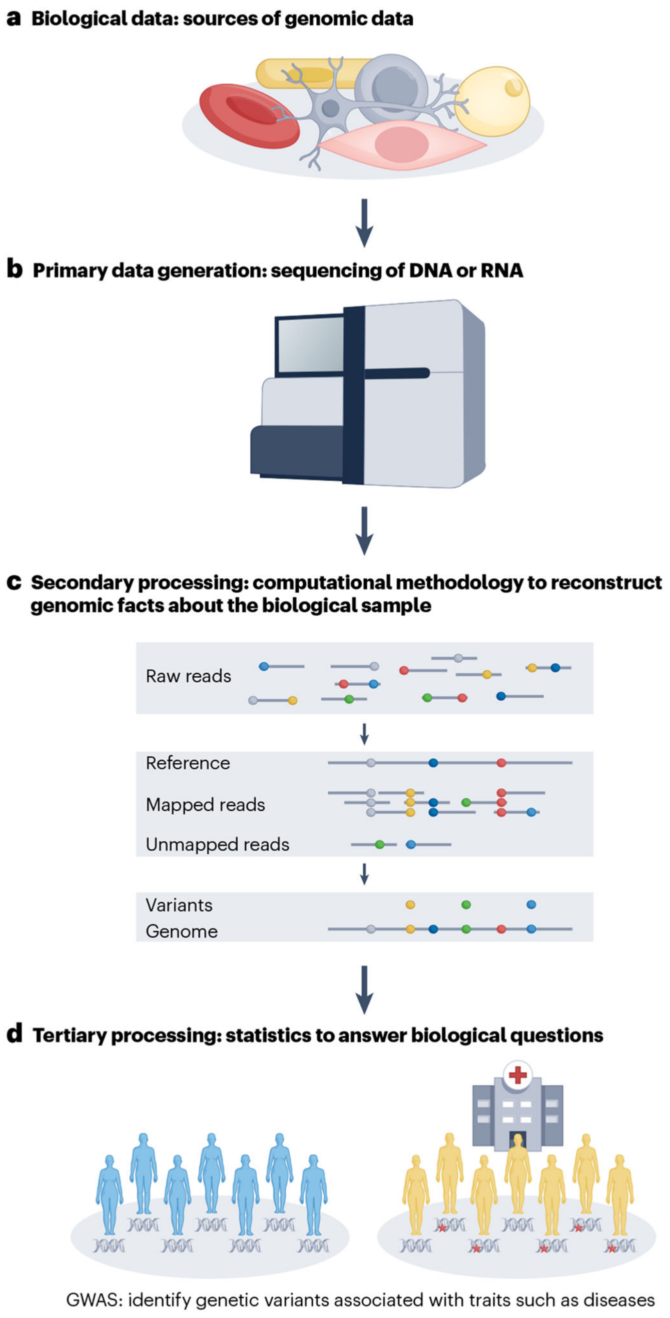

a Biological data: sources of genomic data
b Primary data generation: sequencing of DNA or RNA
c Secondary processing: computational methodology to reconstruct genomic facts about the biological sample
Raw reads
Reference
Mapped reads
Unmapped reads
Variants
Genome
d Tertiary processing: statistics to answer biological questions
GWAS: identify genetic variants associated with traits such as diseases

Fig. 1 |. Overview of genomic analysis pipelines.

a, Biological data sources. Before any analysis takes place, raw data must be gathered. Traditionally, bulk sequencing mixes together DNA or RNA from many cells in a sample — sometimes from a single individual, sometimes from an entire microbiome. More recently, the emerging technology of single-cell `RNA sequencing` (scRNA-seq) allows for collecting and amplifying genomic samples from individual cells. **b**, Chemistry-driven primary data generation (sequencing). A sequencer is a machine that takes a physical sample and outputs digital information in the form of a set of limited-length nucleotide strings called 'reads' (for example, ACGT) with associated metadata. In the early days, Sanger sequencing produced reads of length ~800 bp, but it was the advent of ‘second-generation sequencing’ with reads of ~100–200 bp that have brought costs down sufficiently to produce analytical bottlenecks. Additionally, newer ‘third-generation’ technologies are still expensive, but can access lengths in the >1000 bp range. **c**, Algorithmic secondary processing. Our focus in this Review is on ‘secondary processing’, which is all of the computational methodology used to reconstruct genomic facts about the biological sample from the output of the sequencer — for example, to reconstruct the genome of an individual — but not to specifically resolve biological hypotheses. These secondary processing steps are where the majority of the current computational bottlenecks lie (see Box 2 for more details on the various elements and workflows). **d**, Statistical tertiary processing. ‘Tertiary processing’ uses the output of secondary processing to answer biological questions. For example, determining the genomic variants present in a sample would be secondary processing, but in tertiary processing a researcher may perform a genome-wide association study (GWAS) on those variants to find disease associations. Often, this type of analysis is largely statistical. Although there is substantial computation involved, especially with the rise of deep learning, the computational challenges here are fewer compared with secondary processing.

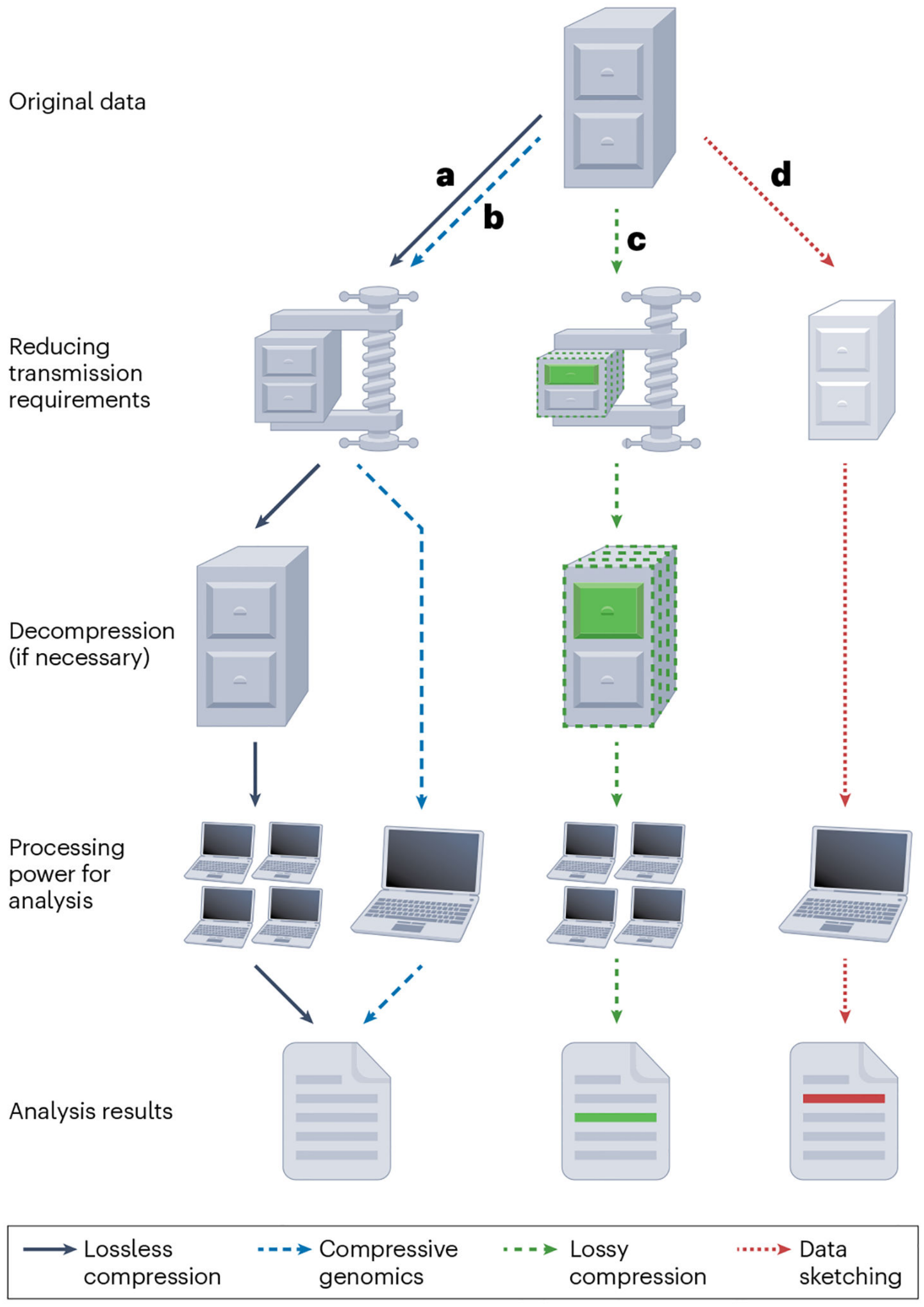


Fig. 2 |. Genomic compression and sketching.
There now exist many different processing techniques for reducing transmission requirements, but they involve different kinds of trade-offs, including reductions in transmission size, decompression requirements, compute time and accuracy of the downstream analysis. **a**, Lossless compression. Data are compressed, creating a smaller squashed version, but the data still need to be decompressed back to the original into order to run computations, which takes a lot of time, as represented by the multiple computers. **b**, Compressive genomics. Data are compressed so as to operate directly on the compressed representation, increasing speed (represented by the single computer for analysis) without

loss of accuracy because the decompression step can be skipped. (Note that there is also a lossy version of compressive genomics, which improves the compression ratio at the cost of accuracy, but here we only illustrate lossless compressive genomics.) **c**, Lossy compression. This method allows for some error in the reconstruction to reduce the compressed file size (depicted in the green modification to the filing cabinet), which may alter the final results of analysis after decompression (hence the green 'error'). **d**, Data sketching. The need for reconstruction can be avoided through a sketch (which completely transforms the data irreversibly, depicted visually as desaturation), but significantly reduces the file size and computation time, which may alter the analysis results (hence the red 'error'). The central difference between sketching and compression is that because compressed data can be decompressed into the same format as the original, the same downstream analysis tools can be used; for sketching, the downstream tools must be modified to operate on sketched data, but sketching can often improve upon space savings over even lossy compression.

Table 1 |

Progression in genome sequencing technologies

Technology	Timeline	Properties	Read length	Errors	Ref.
First generation					
Sanger	Since 1977	Low throughput	1,000bp	0.001% (mixed)	138
Second generation					
Illumina	2007–present	High throughput	2×150bp	0.24% (substitutions)	139
Third generation					
PacBio CLR	2011–present	Single molecule	15–20 kbp	~10% (indels)	140
PacBio HiFi	2019–present	Single molecule	15–20 kbp	~1% (indels)	141
Oxford Nanopore Minion	2016–present	Single molecule; portable	1–200kbp	~5% (indels)	142
Oxford Nanopore Minion Duplex	2021–present	Single molecule; portable	1–200 kbp	~1% (indels)	143
Linking mechanisms					
10× Genomics (now discontinued); BGI Complete Genomics	2015–present	Linked read clouds/synthetic long reads	250 kbp clouds	0.24% substitutions if using Illumina	144
Hi-C	2009–present	Link density (3D structure)	n/a	Depends on sequencing technology	145

BGI, Beijing Genomics Institute; CLR, continuous long read; HiFi, high fidelity; indels, insertions and deletions; n/a, not applicable; PacBio, Pacific Biosciences.

Table 2 |

Examples of modern computational methodologies

Software	Technique	Task	Trade-offs	Refs.
Data compression and compressive software				
SCALCE	Locally consistent encoding	Sequence compression	Increased compression speed and ratio at cost of losing read order	79
ESS-Compress	Spectrum-preserving string sets	On-disc *k*-mer compression	Efficient disk storage at cost of decompression requirement	78
CORA	Homology table	Read mapping	Fast compute at cost of high RAM requirement	71
BEETL-fastq	BWT index	Compressed *k*-mer search	Compressed search at cost of moderately increased RAM requirement	80
BEETL, Quartz, QVZ, Qualcomp, DSRC2 and so on	Lossy quality score compression	Compression of quality scores beyond what is possible losslessly	Much higher compression ratios at cost of giving up exact reconstruction of original data	Survey in ref.[88]
Data sketching (including minimizers, MinHash and so on)				
Mash, BinDash	MinHash sketches	Sequence similarity comparison	Faster comparisons at tuneable cost of accuracy	17,92
MashMap2	MinHash sketches	Homology maps	Faster comparisons at tuneable cost of accuracy	99
Dashing	HyperLogLog sketches	Sequence similarity comparison	Faster comparisons at tuneable cost of accuracy	93
FACS	Bloom filters	Sequence classification	Fast classifications without substantial trade-off	96
BIGSI	Fixed-length binary signatures	Sequence search	Fast search at tuneable cost of accuracy	97
Sequence Bloom Tree	Stacked bloom filters	Sequence search	Fast search at tuneable cost of accuracy	146
mdbg, MBG	Minimizer-space computation	Sequence assembly	Faster assembly without substantial trade-off	101,102
SEDEF	Minimizers	Segmental duplication detection	Faster detection at tuneable cost of accuracy	100
Kraken 2	Minimizers	Taxonomic classification	Fast classification without substantial trade-off	98
Minimap2, Winnowmap	Minimizers	Read mapping	Fast mapping without substantial trade-off	66,103
IsONclust	Minimizers	Transcriptome long-read clustering	Fast clustering, but at potential cost in accuracy	147
Geosketch	Geometric sketching	Single-cell transcriptomic analysis	Better highlights rare cell types at cost of potential erroneous artificial clusters	41
Hopper	Geometric sketching	Single-cell transcriptomic analysis	Provable optimality at cost of somewhat slower runtimes	110
Hardware acceleration				
Illumina Dragen	FPGA	GATK	Hardware acceleration at cost of monetary expense and complexity of set-up	116
Nvidia Clara Parabricks	GPU	GATK		117
SneakySnake	FPGA/GPU	Pre-alignment filtering of reads		113

Software	Technique	Task	Trade-offs	Refs.
GenASM	Custom accelerator design	Approximate string matching for genome analysis	Custom accelerators designed but not manufactured	114
Cloud parallelization				
DNAnexus	Amazon Web Services	GATK	Lowered complexity at cost of monetary expense	
Illumina Dragen AMI	Amazon Web Services	GATK	Lowered complexity of setting up Illumina Dragen at cost of monetary expense	
Domain-specific languages				
BPipe	N/A	Pipeline glue	Reproducible pipeline glue at cost of learning new language	131
Snakemake	N/A	Pipeline glue		132
Galaxy	N/A	Web-based pipeline construction	Easy to use Web platform, but restricted to the (many) tools integrated into the platform	134
SARVAVID	N/A	Genomics primitives	Easy parallelism at cost of learning new language	135
Seq	N/A	Genomics primitives	Fast runtime of compiled language using Pythonlike syntax	72

AMI, Amazon Machine Image; BEETL, Burrows–Wheeler extended tool library; BIGSI, bitsliced genomic signature index; BWT, Burrows–Wheeler transform; CORA, compressive read-mapping accelerator; DSRC2, DNA sequence reads compressor 2; FACS, fast and accurate classification of sequences; FPGA, field-programmable gate array; GATK, genome analysis toolkit; GPU, graphics processing unit; MBG, minimizer-based sparse de Bruijn Graph constructor; mdbg, minimizer-space de Bruijn graphs; N/A, not applicable; QVZ, quality value zip; RAM, random access memory; SCALCE, boosting sequence compression algorithms using locally consistent encoding; SEDEF, segmental duplication evaluation framework.