Lessons learnt on the analysis of large sequence data in animal genomics

Filippo Biscarini[1,4¶*], Paolo Cozzi[1,3], and Pablo Orozco-ter Wengel[2¶]

[1] CNR-IBBA, Milan, Italy

[2] School of Biosciences, Cardiff University, Museum Avenue, CF10 3AX Cardiff, UK

[3] PTP Science Park, Department of Bioinformatics and Biostatistics - Via Einstein, 26900 Lodi, Italy

[4] School of Medicine, Cardiff University, Heath Park, CF14 4XN Cardiff, UK

[¶] these authors contributed equally to this work

Running head:

Analysis of large animal-genomics data

*corresponding author: Filippo Biscarini

Via Bassini 15, 20133 Milan (Italy)

+39 340 7499754

filippo.biscarini@ibba.cnr.it

Summary

The 'omics revolution has made a large amount of sequence data available to researchers and the industry. This has had a profound impact in the field of bioinformatics, stimulating unprecedented advancements in this discipline. Mostly, this is usually looked at from the perspective of human 'omics, in particular human genomics. Plant and animal genomics, however, have also been deeply influenced by next-generation sequencing (NGS) technologies, with several genomics applications now popular among researcher and the breeding industry.

Genomics tends to generate huge amounts of data: genomic sequence data account for an increasing proportion of Big Data in biological sciences, thanks largely to decreasing sequencing costs and large-scale sequencing and resequencing projects.

The analysis of big data poses a challenge to scientists: data gathering currently takes place at a faster pace than data processing and analysis, and the associated computational burden is increasingly taxing, making even simple manipulation, visualization and transferring of data a cumbersome operation. The time taken up by the processing and analysing of huge data sets leaves therefore little time for data quality assessment and critical interpretation. Additionally, when analysing lots of data something is likely to go awry: the software (pipeline, procedure) may crash or stop, and it can be very frustrating to track the error.

We hereby review the most relevant issues related to tackling these challenges and problems, from the perspective of animal genomics, and provide researchers with a framework of steps needed when processing large genomic data sets.

KEYWORDS: big data, genomics, data analysis, next-generation sequencing, animal genetics, ‘omics, computational biology

INTRODUCTION

Big data: these two words have become buzzwords in diverse disciplines. They refer -broadly speaking- to the large quantity of data made available through the extraordinary technological improvements in the automated collection of information (Lohr, 2012). Big data have brought about a whole new epistemology, leading to the emergence of a fourth paradigm in science (Hey et al. 2009, Bell, 2009; Kitchin, 2014), that is, after theoretical, experimental and simulation science, it is now the era of data-driven science. This revolution is impacting several fields of science, including bioinformatics (Schuster, 2008; Pop and Salzberg, 2008): e.g. the European Bioinformatics Institute (EBI) stores over 60 petabytes (60 x 10^{15} bytes) of data, of which over 2 petabytes are genomic data (Marx, 2013); the Sequence Read Archive (SRA) at the National Centre for Biotechnology Information (NCBI) contains more than 3.6 petabases of data (4 bases ≊ 1 byte). Table 1 gives examples of large ‘omics data.

Genomics is no longer an emerging field but an established one, which is projected to be among the domains of science and technology that will generate the largest amounts of data by 2025 (Stephens et al. 2015), largely as a consequence of falling sequencing costs (Figure 1). Animal genomics accounts for an increasing proportion of this amount, thanks also to large-scale sequencing and resequencing projects such as the 1000 bull genomes project (http://www.1000bullgenomes.com/), or the EU’s FP7 Nextgen project (http://nextgen.epfl.ch/) among others. Genomic selection 2.0 is potentially another source of large amounts of sequence data in livestock (Hickey, 2013). The challenge represented by the analysis of big data in animal genetics has been already recognized by the scientific community (e.g. Cole et al., 2011; Tempelman, 2016; Perez-Enciso, 2017): data gathering has currently a faster pace than data processing and analysing; the associated computational burden is increasingly taxing, making even simple manipulation, visualization and transferring of data a cumbersome operation; the time taken up by the processing and analysing of huge data sets leaves little time for its critical interpretation; when analysing lots of data, something is likely to go awry, the software, pipeline or procedure may crash, or stop, and it can be very frustrating to track the error.

Here we review the most relevant issues related to the analysis of large sequence data in animal genomics. Additionally, we propose some useful guidelines to tackle these challenges and problems, and provide researchers with a framework of steps needed to face the processing of large sequence experiments. These indications were motivated by research work with large sequence data from livestock genomics experiments; the framework however, applies equally well to non-livestock animal, plant and human genomics (and, more generally, to the analysis of big “omics” data). For the sake of illustration, we will refer all-along to a standard mammalian genome organized in chromosomes, and a setting in which several animals (individuals) are sampled. Before starting off through this review, we kindly remind the reader of a basic principle: always conceive effective algorithms and write efficient scripts for your data analysis!

PRELIMINARY CHECKS AND PLANNING

The internet is a very large resource providing links to publications, software download sites, databases and others. However, navigating this forest of options can be difficult and discouraging, resulting in researchers opting for developing tools that enable them answering the questions of immediate pertinence to their work. Usually, the development of such tools requires the knowledge of programing skills (e.g. C++, Java, Python, R), which still today are not part of the standard toolkit of life science researchers (Ditty et al. 2010; Mangui et al., 2017). Developing programing skills is very valuable in terms of i) widening the range of questions that can be tackled by removing the dependency on available software, ii) the applicability of programing skills beyond the immediate area of research, iii) reproducibility of research results, and iv) transferable skills. However, a lack of acquaintance with the available online resources can result in the inevitable re-invention of the wheel.

As pointed out already by Osborne et al. (2014), the first question that needs addressing is whether your "question of interest" has already been asked and, especially, answered. Online databases can help solving this issue by providing access to the literature (e.g. Pubmed, Scopus or the Web of Science, Google Scholar), data (e.g. Genbank, Ensembl), and software (e.g. Sourceforge - http://sourceforge.net/ - and Github - https://github.com/). Secondly, what are the resources available to answer the question of interest? A plethora of online resources for genomics already exists, e.g. repositories of gene annotations, SNP (single nucleotide polymorphism) and other variants, as well as cross species comparisons for genomic regions of interest, such as Ensembl (www.ensembl.org), or the UCSC Genome Browser (https://genome.ucsc.edu/). Many of these online resources also host up-to-date genome reference sequences and annotations that can be used to compare the data produced by researchers for quality purposes. Third, researchers "are not alone" and are not likely the first to face a particular problem. Beyond these resources, several online portals open the possibility for both experienced and inexperienced researchers to exchange knowledge in the form of question-and-answer forums. SEQanswers (http://seqanswers.com/) and Biostars (https://www.biostars.org/) are community driven forums of users focused on the discussion of next-generation genomics related issues ranging from technology development to bioinformatics support, and biological data analysis. ResearchGate (www.researchgate.net) hosts a large community of researchers from diverse disciplines to archive, disseminate and discuss scientific publications, ask and answer questions, propose and comment research projects and ideas. Lastly, but not of least importance, Stack Overflow (http://stackoverflow.com) and Stack Exchange (http://stackexchange.com/) are similar users portals, but which exclusively focus on statistics, programming and computing related issues, with extensive archives on discussions on both general and specific issues, covering most of the standard computing languages used in life sciences (e.g. Python, Java, R). Additionally, traditional peer reviewed articles offer further guidelines on software, data analysis and best practices, e.g. Nicolazzi et al. (2015) provided a review of currently available software solutions for researchers working in this field, and tools to streamline the analysis of animal sequence data are constantly being released (e.g. the Zanardi suite, Marras et al. 2016; Consesa et al 2016). Table 2 summarizes some of the publicly available resources.

Large sequence data not only comprise the millions of reads (i.e. sequences) from next generation sequencing platforms, but other data types too, like large scale genotyping data (e.g. high density SNP arrays with hundreds of thousands of genotypes for thousands of individuals, such as in genomic selection programmes: e.g. Van Raden et al 2011; Meuwissen et al., 2016). The data deluge unleashed by “data-driven” biology can easily become overwhelming (Hawkins et al. 2010; Berger et al. 2013). This problem arises from two main issues related to handling this type of data. The first one is the sheer size of the data, e.g. the amount of space required to store the data, work with it (temporary storage) and archiving it to guarantee its availability in the future. To give an idea, the complete genome of a single bovine is about 20-40 GB in size, in terms of (compressed) raw sequence data. Researchers need to assert the size of the data that is expected they will receive from an experiment, and accordingly purchase the hard-disk space necessary to maintain it, ensuring there is enough working memory (RAM) to handle the data, plentiful temporary space where intermediate files of multiple analyses can be stored. Additionally, the data should be backed up regularly, and ideally it should be available to all users at all time, e.g. via a server with a mirrored system that can be accessed online via secure shell or other protocol. While many researchers can purchase space/time in a local server clusters, others have to opt for online alternatives (e.g. cloud-based computing). Whatever the choice is, researchers need to carefully consider the additional budget necessary for such venture as the price per Tb of space is still expensive despite of the continuous fall of the price per byte and personal computers and laptops do not tend to be powerful enough.

The second issue deals with a change in paradigm of handling the data. Until not so long ago researchers were used to scrupulously look at each piece of data, back up all intermediate steps of data analysis, transferring files between storage locations using flash drives or even hard drives. However, typical dataset sizes in this era are easily hundreds of Giga bytes (Gb) large, if not Tera bytes (Tb) or more (Schadt et al. 2010). Consequently, a new paradigm must be defined where data can be i) efficiently summarised in order to identify approaches to trim it (e.g. remove data of lower quality and thus less reliability), ii) avoid unnecessary backing up intermediate analysis steps that are not crucial, as these can rapidly increase the total data size by orders of magnitude, iii) avoiding unnecessary transfer of data between locations, as data can take days or hours to transfer using internet protocols, and iv) carefully document the steps taken at all stages of data analysis (i.e. write down an analysis pipeline) for reproducibility purposes. In other words, be pre-emptive and estimate data size and its associated costs, and be tidy by keeping track of all analyses applied with master scripts and copies of the software used to handle data. For instance, the National Institutes of Health (NIH) is developing the Big Data to Knowledge initiative (BD2K), that aims at managing large dataset in biomedicine, with elements such as data handling and standards, informatics training and software sharing (Marx 2013). Without these considerations researchers won’t have enough space or RAM for analyses, and very importantly, researchers won’t be able to reproduce results contributing to the endless list of unreproducible published data (Nekrutenko & Taylor 2012).

COMPUTING INFRASTRUCTURE AND BASIC REQUIREMENTS

The advent of large genomics datasets brought about computational challenges which relate to the available computing infrastructure. A *de novo* genome assembly requires approximately 1 Gb of RAM for every 1 Mbps of genome, which for the bovine genome (~2.7 Gbps) would translate to at least 3 TB of available RAM. Traditionally, larger problems were addressed by scaling-up i.e. resorting to supercomputers with several processing units and large RAM capabilities (e.g. a petaflop supercomputer for protein 3D-folding, Allen et al. 2001). This solution can be very fast for medium scale problems, but it requires highly specialized software which tends to be very expensive. Additionally, with ever increasing size of the data, this approach would eventually hit a wall.

Scaling-out to using a network of machines is an appealing alternative. One option are high performance computer clusters, typically constituted by a number of good quality computing machines accessible through a local connection like an organization's intranet. An example is the bioinformatics computing facility at PTP Science Park (www.ptp.it), with over 700 cores and 3.5 TB of memory. Computer clusters are generally high performing and comprise homogeneous machines, which make it easier to distribute programming over the network. Downsides are the expensive maintenance and the frequent underutilization: the need for very large computations in any given organization is typically not continuous, but "bursty" in nature.

Computer clouds are an alternative option for distributed computing, which may circumvent some such limitations. Cloud-based infrastructure services build on commodity hardware, individually cheap, which is assembled into very large networks capable of scaling to massive computation problems. Commercial services on a pay-per-use basis are attractive since they permit to avoid investing in infrastructure and maintenance, and limit costs to the actual calculations that are needed. Examples of such services are Amazon Web Services, HPCloud, Google Compute Engine, Windows Azure: this market is changing rapidly, and is finding applications also in genomics (O'Driscoll et al. 2013). Major challenges in cloud computing are usually represented by network communication and by the additional software complexity generated by dealing with heterogeneous hardware. This can be handled through frameworks for distributed computing like Apache Spark (Meng et al., 2015), implemented in platforms such as DataBricks (https://databricks.com/).

Distributed computing is certainly the way to go for animal genomics, be it private computer clusters or commercial public cloud services. A pre-requisite is generally to work on a Unix/Linux environment, although virtualization technology allows access also to Windows users (Krampis et al. 2012).

DATA STORAGE: DATABASE & CO.

The amount of data generated by genomics is huge, and projected to be enormous: Stephens et al. (2015) determined that over 100 PB of storage are currently used by the 20 largest sequencing institutions, and estimated that as many as 40 EB (exabytes - 10^{18}) of storage capacity may be needed by 2025. These requirements may be partially alleviated by data compression (Loh et al.

2012) or through techniques like “delta encoding” (Christley et al. 2009), by which only variants are stored instead of complete genome sequences, at least for some individuals.

High-density genotyping and sequence data are often distributed as ASCII or binary files. Such files need however to be parsed each time you need to access even a subset of the data, thereby making the analysis quite cumbersome. While the availability of data files in standard formats is usually an excellent option (e.g. VCF or BAM files have become a standard in genomics), these files may be enormous making data handling cumbersome. An alternative are relational databases, which offer more efficient ways of storing, accessing, extracting and analysing data in a neater and safer manner. Data in a relational database are represented in tables linked through unique record IDs, and are processed with SQL (structured query language), a programming language specifically designed to handle data and their relations. Building a full relational database (e.g. mySql) is an ideal choice for long-term storage and maintenance of data. However, such databases may be complex and time- and resources-consuming, as they rely on client/server applications, and most of the times the server-side component need to reside on a dedicated infrastructure accessible over a network to guarantee scalability and availability. However, for smaller projects, simpler solutions like sqLite exist (https://www.sqlite.org). SqLite allows making use of ordinary files to store data and their relations using a transactional model, instead of building a client/server database. Such files are portable across platforms and besides storing data, they also encode high-level functionalities (e.g. “Application File Format”, like MS Excel, Epub or Pdf files). However, this flexibility does not come without a cost: for instance, when multiple applications or users need to read/write data at the same time (concurrency), or increasing network operations is desirable (e.g. to generate and record results), or scaling-up has to be dealt with, SqLite would not be sufficiently performing, and a full server/client approach has to be considered instead.

Relational databases, both with a database server or in the no-frills sqLite version, are very powerful tools that need the tables describing the data to be adequately indexed in order to make efficient use of them. On one hand, without an index, if a specific row is queried the relational database management (RDBM) system performs a sequential scan row by row in the table to check whether its name attributes match our query conditions; the speed of such sequential search is proportional to the number of rows in the table, i.e. it is *O(N)* implying that the number of operations required is the number of rows (N) in the table. However if the database is indexed instead, the scanning speed is *O(log(N))* (for the default B-tree index type; Owens, 2006), because only the index needs to be accessed by RDBM. An index is a specialized data structure that stores the values for one or more columns in the database tables in a highly optimized way. Additionally, indexing is even more relevant when joining tables, as that enables matching rows on each table that have the same key, instead of having to sequentially scan each pair of tables using a total of *O(N*M)* operations (where N and M are the numbers of rows in each table). On the other hand, indexes are data structures that take up more space than default attributes (i.e. table columns), and that need to be maintained by the RDBM when records are modified. Therefore, indexing too many table columns would *i)* be a waste of resources and *ii)* cause an overall performance degradation. Consequently, identifying the right descriptors to be used in indexes is crucial, and requires taking into account the cardinality of the data and anticipating the most common and suitable queries of the database. For example, when querying sample genotypes on a chromosomal sequence it would make no sense to index records on the sample sex attribute (male/female), given its low cardinality; instead, the position of a polymorphism along the genome would make a good index, allowing accessing a reduced set of rows upon query.

Recently, innovative database architectures are emerging, such as graph databases, which hold the promise of better modelling highly interconnected data like for instance computer networks. Storage and querying such data in graph databases are expected to be faster and, in general, more efficient (Angles and Gutierrez, 2008). Interconnected data in animal genetics may be illustrated by genealogies (animals as nodes and relationships as connections), phenotypic records (traits as nodes and trait-animal connections as trait values) and SNP genotypes (SNP loci as nodes and SNP genotypes for individual animals as connections; see Biscarini et al., 2013b, for an example).

DATA ANALYSIS

The analysis of genomic data may be very diverse, depending on the objective: this may go from de novo assembly of a genome, to sequence alignments and variant calling; or may be the downstream statistical analysis of genomic data, such as phylogenetic studies, genome-wide association studies or genomic predictions for phenotypes of interest in animal breeding (e.g. de los Campos et al. 2013). For large problems involving vast sequence data for a large number of individuals (e.g. hundreds of thousands of genotyped animals like the US Holstein cattle population), scalability is certainly an issue, and a distributed computation setting on a computer cloud or cluster is needed. Frameworks to run the analysis over a network of machines are used to first distribute the computations to where the data reside (Map operation) and then aggregate results at the end (Reduce operation). Google MapReduce is one such solution to process big data (Taylor 2010), which can be effectively coupled with machine learning algorithms for the analysis of large datasets (e.g. Gillick et al. 2006), by resorting for instance to linear algebra techniques like inner and outer products between distributed matrix rows and columns, or to feature-encoding techniques like one-hot encoding or feature hashing. Machine learning is becoming increasingly popular in genomics (e.g. Szymczak et al 2009) and in animal breeding (e.g. Gonzalez-Recio & Forni 2011). A popular combination is given by the scripting language Python within the Apache Spark framework for distributed computing (Meng et al. 2016).

Another recent and productive line of research is to develop “streaming” or “online” algorithms that can analyze data on the fly without the need of storing it all in memory. Two examples are the Sailfish (Patro et al. 2014) and Kallisto (Bray et al. 2016) quantification algorithms for reads from RNA sequencing experiments, that are orders of magnitude faster than standard approaches while presenting similar or superior accuracy. Such approaches are currently applied to ‘omics technologies other than genomics, but it can be envisaged that similar ideas may soon find application also for the analysis of large genomic datasets.

Open-source projects like Galaxy (https://galaxyproject.org/) and Jupyter (http://jupyter.org) offer sophisticated platforms for data analysis which ease entry barriers for comparatively less programming-savvy life-science researchers (Grüning et al., 2017).

Big data are not only large in size but also tend to be heterogeneous in nature: in genomics, one may think of different sources (SNP-arrays, RAD-sequencing/Genotyping-by-sequencing, whole-genome sequences), different genome assembly or array design and density, gene annotations data, and so on (Perez-Enciso, 2017). Heterogeneous data pose challenges for data integration and for imputation of missing values, and may harbour a certain amount of noise (errors) which should be taken into account when analysing the data (Pompanon et al., 2009; Biscarini et al., 2016; Biffani et al., 2017).

WRITING CODE AND RUNNING THE ANALYSIS

The increasing availability of multiple-core computers and computing clusters with several processing units (CPUs), has prompted the use of parallel computing, where large problems can sometimes be divided into smaller ones that are distributed over hundreds of CPUs and solved concurrently ("in parallel") improving execution times. The analysis of sequence data often present embarrassing parallel problems: e.g. genome sequences can be analysed per chromosome, or alignments can be performed on a per sample (and per chromosome) basis (see for instance Sikorska et al., 2013). Embarrassing parallel problems are “embarrassingly” easy to run in parallel, e.g. the user just needs to split the job into sub-jobs and run them independently on different cores/CPUs/machines. In such cases, the computation time is a direct function of the processing resources (n. of machines, processing units such as in Beowulf clusters). Parallelization may though be less straightforward when sub-processes are not thoroughly independent and some degree of communication between them is needed to achieve the final solution. When such communication is minimal, we talk of “coarse-grain” parallelization: an example is algebraic matrix inversion frequently used in genetics and genomics (e.g. Biscarini et al., 2013a). Sometimes though, sub-processes need to communicate extensively by sharing memory, coordinating I/O, or reciprocally update intermediate values. Such fine-grain parallelization problems are more difficult to implement and run in parallel, and require the design of clever algorithms. Examples of fine-grain parallelization with sequence data are the GPU-Blast implementation of the Blast alignment algorithm (Vouzis and Sahinidis, 2011), and the determination of progressive alignments topology in the clustalW algorithm (Li KB 2003).

Interpreted scripting languages have many useful features that facilitate the execution of complex tasks. For instance, R (R Core Team, 2013) can implement complex statistical models; or, high-level scripting languages like Python (Van Rossum & Drake, 1995) allow to execute complex tasks with just a few lines of easy-to-read code. Compiled languages like C/C++ or Fortran, on the other side, achieve higher computing performances and a more powerful memory management, because they translate directly to the native code of the specific machine. The latter, however, comes at the expense of easy implementation, since compiled languages typically use low level functions and very simple data structures that force users to write extensive code even for relatively simple tasks. Hybrid solutions between compiled and interpreted languages that improve computational performances with no need of sacrificing the user-friendly syntax of scripting languages exist. Examples include Cython (Behnel et al., 2011), SWIG (Beazley et al., 1998), the Rcpp R library (Eddelbuettel & François, 2011), that offer frameworks where users can identify and implement in a compiled language only the bottlenecks of their algorithms, while keep writing everything else in an interpreted user-friendlier language. Such hybrid schemes provide therefore a compromise between performance and complexity. Based on our experience, embedding Cython blocks in a script allowed processing 0.5 Gb of sequence data in 50.380 seconds compared to 207.266 seconds with the same algorithm solely implemented in Python (*ceteris paribus*).

Modular programming refers to the organization of the code in subunits which act more or less independently (Maynard, 1972). Organising the code in modules or functions (or classes, in the object-oriented paradigm) is especially useful for complex programmes or pipelines that comprise several tasks, entail a considerable running-time, or run extensively in parallel. Modularity allows for the code to be recycled -functions, modules or classes are typically used repeatedly- and portable across platforms or projects (no need of re-writing everything from scratch each time), and is a key component of programming efficiency. Besides, modular code is easier to debug, since you can conveniently go through the program/pipeline "piece by piece", and allows to track even problems independent from your code, like machine or cluster breakdowns, electric network failures etc ...: you would be able to resume the work from where the problem occurred and relaunch only what is really needed, instead of everything from start. This makes your pipeline more robust to system crashes, and reduces the risk of losing data. A well known example of a modular pipeline of analysis for sequence data is the Ensembl pipeline for the annotation of genomic sequence (Potter et al., 2004). To recap, make your code modular and you'll have an array of advantages, at the expense of only little extra planning effort!

Once you have made your code/pipeline modular, you need to make sure it is reproducible. This can be achieved by organizing it into e.g. R packages or Python modules. Or it can be organized into a reproducible pipeline making use of a data/analysis serialization format like the XML mark-up language, the INI format or YAML. This latter, YAML (recursive acronym: Yaml Ain't Mark-up Language), has the advantage of being human-readable and of having an easy syntax suitable for all programming languages (Ben-Kiki et al., 2005). YAML helps dealing with big data projects with several parameters and jobs to be launched independently. It is useful to handle the serial steps of a pipeline, but is particularly suited for "embarrassing parallel" problems, where besides running several consecutive steps, these are to be repeated over a large number of samples. A modular pipeline plus YAML serialization format is a powerful combination for the analysis of large sequence data. YAML is usually organised in two files, one with the serial steps of the analysis, the other with the samples over which the analysis should be run in parallel (see Box 1 for an illustration). YAML files are written as hash tables/associative arrays, i.e. in the form of key-value pairs. YAML syntax is overly simple: the most important rules to remember are indentation, a few keywords (e.g. resources, steps, samples) and placeholders (i.e. <variable_name>). In order for the analysis to be run, YAML files need to be interpreted by ad hoc programmes/scripts, like for instance the PipEngine launcher developed in Ruby (Strozzi & Bonnal, 2017).

Box 2. How YAML works in practice

For bioinformatics tasks, typically the YAML data analysis serialization format comprises two files (.yml): 1) “configuration file” listing resources (paths to input data and output directories) and samples to run the analysis in parallel; 2) “analysis file” describing the serial steps of the analysis and related resources (programmes, scripts). YAML files are written in the form of hash tables/associative arrays: ‘key’: value. Below an illustration for the SNP calling and missing genotype imputation over 100 samples.

```
#-----------------------
# configuration.yml
#-----------------------
resources:
       output: /output/directory/
       data: /path/to/data

samples:
       ‘sample1’: sample1_name
       ‘sample2’: sample2_name
       ……
       ‘sample100’: sample100_name


#-----------------
# analysis.yml
#-----------------
resources:
      snp-calling_program: /path/to/snp-calling_program
      imputation_program: /path/to/imputation_program

steps:
       snp-calling:
              desc: call snps from each sample sequence
              run:  <snp-calling_program> --input <sample> -o called_snp.<sample>
              cpu: 4

      imputation:
              desc: impute missing genotypes at SNP loci
              run:  <imputation_program> --input called_snp.<sample> --output
imputed_snp.<sample>
              cpu:  4
```

In this simple example, the steps of the analysis are organised with a description of the step, the actual code to be run in each step, and the number of CPU to be used. The analysis can then be run through and ad hoc interpreter (see main text) using a command line similar to the following:

```
>> pipengine run --pipeline analysis.yml --samples-file configuration.yml --name
imputation --steps imputation
```

Processing data loaded onto the (volatile/RAM) memory is much faster compared to the heavy workload of repeated I/O operations involved in reading stored data and writing them back out on the disk (exactly how faster depends on disk and memory architecture: e.g. SSD, HDD, DDR3). When analysing relatively small datasets, this is usually not a problem, even on a laptop/client PC: all the data can be placed in the memory and analysed efficiently from there. With large sequence data this is often not possible, not even if large RAM capacities are available as in computing clusters or high-performance servers. This is especially true when not just a single “large” job has to be executed, but several parallel jobs are to be run simultaneously and have to compete for memory resources: if several “large” jobs are launched in parallel, the memory would soon be full! In such cases, CPU-intensive rather than memory intensive computing strategies should be adopted: the software would thus need to be designed so to resort as much as possible to I/O operations in order to reduce the memory burden. Data can be read in the memory record by record, or in chunks, and then processed by the CPU. In such a setting, there is a trade-off between memory usage and CPU-time: memory efficiency is gained at the expense of increased computation time (repeated I/O operations). An illustration from sequence data is for instance reading FASTA files: these are usually quite big files, and loading them into memory would easily exhaust memory resources. It makes therefore sense to read such files sequentially, which won't use much system memory. In some circumstances, though, repeated access to (part of) the file is needed, like in most matrix operations: then the approach of reading the whole file into memory makes the algorithm much easier to write, at the cost of some system memory.

PUBLISHING RESULTS, DATA AND CODE

In the previous sections we attempted to emphasise that researchers working on large datasets usually encounter problems that are very similar, and which in many cases others also have encountered and frequently solved. It is possible to gain access to that communal knowledge by querying the literature, public databases, open forums and discussion groups. In the same way, it impends on researchers to make their knowledge publicly available as members of the "scientific community" (Budd et al. 2015). For that purpose it is important to identify the public databases where raw data used for research can be stored. Such approach serves two purposes. On one hand prevents researchers from having to come up with the funds necessary to secure data archiving and iis availability in the future (i.e. public databases are free). On the other hand, by using public databases researchers make sure that their work contributes to the continuous growth of the scientific community. Depending on the type of data, several public repositories are available, e.g. DRYAD (http://datadryad.org/), Zenodo (https://zenodo.org/), the Short Read Archive (NCBI, http://www.ncbi.nlm.nih.gov/sra), the European Nucleotide Archive (EBI, http://www.ebi.ac.uk/ena).

While publishing the data used for analyses and the metadata associated to it is a very important step, publishing the analyses pipelines (i.e. the collections of bioinformatics scripts used) is crucial, and regretfully, still rarely done (Ince et al. 2012). Several public repositories exist that enable publishing scripts used for data analysis, e.g. Google Code (https://code.google.com/), Sourceforge (http://sourceforge.net/), Github (https://github.com/) or GitLab (https://about.gitlab.com/). The users community expects to find in this type of repositories scripts that can be directly used by others; however, researchers frequently write code that was intended for their own use or for a specific task (a.k.a. quick and dirty script). While publishing those scripts is still important, programing skills are no longer a desired skill only for mathematicians, physicists and engineers: researchers in the biological sciences, too, need to build a basic informatics knowledge (Dudley & Butte 2009, Hawkins et al. 2010) that enables them writing scripts that are accessible to others (i.e. that can be read and modified).

Finally, although researchers plan their work so to maximize the likelihood of obtaining significant and relevant results, it is of fundamental importance to also publish lack of or negative results, so to minimize issues with publication and reporting bias (Dwan et al., 2008): on-line archives like Bioarxiv (http://www.biorxiv.org/) offer a convenient way to make all research results readily available to the scientific community and the broader public.

CONCLUSIONS

The advancement in 'omics technologies has guided the development of a data-driven approach to biological sciences. This change has marked the need for researchers in the biological sciences to change their approach to experiment design, data handling and storage, and time allocation for wet-lab vs. dry-lab (computer based) work, as well as it has resulted in the growing need for those researchers to at least have a basic understanding of computing language (e.g. to at least be able to look at files) and information technology (e.g. to understand about file transfer protocols between servers). Fast computers and vast storage capabilities are giving us plenty of possibilities to handle large-scale data (besides contributing to produce big-data, in a sort of virtuous/vicious cycle). However, such resources, though ample, are not infinite, and the design of good computation strategies is still fundamental to handle today's large quantities of data. In this review of common practices we described principles that we feel are very important and that biological researchers embarking in the field of genomics need to be aware of. Importantly, while our views derive from our experience working with livestock genomics, our comments are equally applicable to research on crops, wildlife fauna and flora, humans, and microbial 'omics technologies. Lastly, these comments reflect the lessons we learnt during our own experience, and it is very important to note that no matter how well you plan experiments and how strictly you follow our guidelines , when analysing large data involving multiple comparisons, methods, models, samples etc, you must be patient and willing to learn at each step, as you cannot expect that everything will run smoothly without problems!

Acknowledgements

The authors acknowledge the contribution of the NEXTGEN FP7 EU and ClimGen projects from which they received financial support and the opportunity of experimenting with large genomic data. A special thanks/acknowledgement goes to Ian Streeter for his scientific and technical support in dealing with large sequence data. FB was financed also by the Marie-Curie European Reintegration Grant NEUTRADAPT.

Conflicts of interests

The authors declare that they have no conflicts of interests.

REFERENCES

Allen F., Almasi G., Andreoni W., Beece D., Berne B.J., Bright A. et al. (2001) Blue Gene: a vision for protein science using a petaflop supercomputer. *IBM systems journal* **40**(2): 310-27.

Angles R, Gutierrez C. (2008). Survey of graph database models. *ACM Computing Surveys (CSUR)*, **40**(1): 1.

Beazley DM (1998). Interfacing C/C++ and Python with SWIG. In*7th International Python Conference, SWIG Tutorial*.

Behnel, S., Bradshaw, R., Citro, C., Dalcin, L., Seljebotn, D. S., & Smith, K. (2011). Cython: The best of both worlds. *Computing in Science & Engineering*, *13*(2), 31-39.

Bell G, Hey T, Szalay A (2009) Beyond the data deluge. *Science* **323**: 1297-1298.

Ben-Kiki O, Evans C, Ingerson B (2005). YAML Ain't Markup Language (YAML™) Version 1.1. *yaml. org, Tech. Rep*.

Berger, B., Peng, J., & Singh, M. (2013). Computational solutions for omics data. *Nature reviews. Genetics*, *14*(5), 333.

Biffani, S., Pausch, H., Schwarzenbacher, H., & Biscarini, F. (2017). The effect of mislabeled phenotypic status on the identification of mutation-carriers from SNP genotypes in dairy cattle. *BMC research notes*, *10*(1), 230.

Biscarini F, Picciolini M, Stella A, Iamartino D, Strozzi F (2013a) A graph database to store and manage phenotypic, pedigree and genotypic data of livestock. Book of abstracts No. 19 of the 64th Annual Meeting of the European Federation of Animal Science (Nantes, France).

Biscarini F, Pedretti A, Ober U, Erbe M, Jorjani H, Nicolazzi E, Picciolini M (2013b) Use of molecular markers to estimate genomic relationships and marker effects: computation strategies in R. In: The R User Conference, useR! 2013 July 10-12 2013 University of Castilla-La Mancha, Albacete, Spain (Vol. 10, No. 30, p. 13).

Biscarini, F., Nazzicari, N., Broccanello, C., Stevanato, P., & Marini, S. (2016). “Noisy beets”: impact of phenotyping errors on genomic predictions for binary traits in Beta vulgaris. *Plant methods*, *12*(1), 36.

Bray NL, Pimentel H, Melsted P, Pachter L (2016) Near-optimal probabilistic RNA-seq quantification. *Nature biotechnology* **34**(5): 525-527.

Christley S, Lu Y, Li C, Xie X (2009) Human genomes as email attachments. *Bioinformatics* **25**(2): 274-275.

Cole J, Newman S, Foertter F, Aguilar I, Coffey M (2012) Breeding and genetics symposium: Really big data: Processing and analysis of very large data sets. *Journal of Animal Science* **90**: 723–733.

Conesa A, Madrigal P, Tarazona S, Gomez-Cabrero D, Cervera A, McPherson A, Szcześniak MW, Gaffney DJ, Elo LL, Zhang X, Mortazavi A (2016) A survey of best practices for RNA-seq data analysis. *Genome biology* **17**(1): 1.

de los Campos G, Hickey JM, Pong-Wong R, Daetwyler HD, Calus MP (2013) Whole-genome regression and prediction methods applied to plant and animal breeding. *Genetics* **193**(2): 327-345.

Ditty J, Kvaal C, Goodner B, et al. (2010) Incorporating genomics and bioinformatics across the Life Sciences curriculum. *PLoS Biology* 8:e1000448

Dwan, K., Altman, D. G., Arnaiz, J. A., Bloom, J., Chan, A. W., Cronin, E., ... & Ghersi, D. (2008). Systematic review of the empirical evidence of study publication bias and outcome reporting bias. *PloS one*, *3*(8), e3081.

Eddelbuettel, D., François, R., Allaire, J., Chambers, J., Bates, D., & Ushey, K. (2011). Rcpp: Seamless R and C++ integration. *Journal of Statistical Software*, *40*(8), 1-18.

Gillick D, Faria A, DeNero J (2006) Map-reduce: Distributed computing for machine learning. Berkley 18.

González-Recio O, Forni S (2011) Genome-wide prediction of discrete traits using Bayesian regressions and machine learning. *Genetics Selection Evolution* **43**(1):1.

Grüning, B. A., Rasche, E., Rebolledo-Jaramillo, B., Eberhard, C., Houwaart, T., Chilton, J., ... & Nekrutenko, A. (2017). Jupyter and Galaxy: Easing entry barriers into complex data analyses for biomedical researchers. *PLOS Computational Biology*, *13*(5), e1005425.

Hey T, Tansley S, Tolle K (2009) *The Fourth Paradigm: Data-Intensive Scientific Discovery*, Redmond, WA: Microsoft Research.

Hawkins RD, Hon GC, Ren B. (2010) Next-generation genomics: an integrative approach. *Nature Reviews Genetics* **11**(7): 476-486.

Hickey JM. (2013) Sequencing millions of animals for genomic selection 2.0. *Journal of Animal Breeding & Genetics* **130**(5): 331-332.

Kitchin R (2014) Big Data, new epistemologies and paradigm shifts. *Big Data & Society* **1**: 1-12.

Krampis K, Booth T, Chapman B, Tiwari B, Bicak M, Field D, Nelson KE. (2012) Cloud BioLinux: pre-configured and on-demand bioinformatics computing for the genomics community. *BMC Bioinformatics* **13**(1): 1.

Li KB (2003) "ClustalW-MPI: ClustalW analysis using distributed and parallel computing", *Bioinformatics* **19**(12): 1585-1586.

Loh PR, Baym M, Berger B. (2012) Compressive genomics. *Nature biotechnology* **30**(7): 627-630.

Lohr, S. (2012). The age of big data. *New York Times*, 11.

Marx, V. (2013). Biology: The big challenges of big data. *Nature* **498**(7453): 255-260.

Mangul, S., Martin, L. S., Hoffmann, A., Pellegrini, M., & Eskin, E. (2017). Addressing the digital divide in contemporary biology: Lessons from teaching UNIX. *Trends in Biotechnology*.

Maynard, J. (1972) "Modular programming." CRC Press

Meng, Xiangrui, et al. (2016) "Mllib: Machine learning in apache spark." *Journal of Machine Learning Research* **17(**34): 1-7.

Meuwissen, T., Hayes, B., & Goddard, M. (2016). Genomic selection: A paradigm shift in animal breeding. *Animal frontiers*, *6*(1), 6-14.

“NEXT GENERATION METHODS TO PRESERVE FARM ANIMAL BIODIVERSITY: NEXTGEN” FP7-EU Project [http://nextgen.epfl.ch/]

Nicolazzi EL, Biffani S, Biscarini F, Orozco ter Wengel P, Caprera A, Nazzicari N, Stella A. (2015) Software solutions for the livestock genomics SNP array revolution. *Animal genetics* **46**(4): 343-353.

O’Driscoll A, Daugelaite J, Sleator RD. (2013) ‘Big data’, Hadoop and cloud computing in genomics. *Journal of Biomedical Informatics* **46**(5): 774-781.

Osborne JM, Bernabeu MO, Bruna M, Calderhead B, Cooper J, Dalchau N et al. (2014). Ten simple rules for effective computational research. *PLoS computational biology* **10**(3): e1003506.

Owens M (2006) “The Definitive Guide to SQLite”, Chapter 4, pp 155-158, isbn: 978-1-59059-673-9

Patro R, Mount SM, Kingsford C (2014) Sailfish enables alignment-free isoform quantification from RNA-seq reads using lightweight algorithms. *Nature biotechnology* **32**(5): 462-464.

Pérez-Enciso, M. (2017). Animal Breeding learning from machine learning. *Journal of Animal Breeding and Genetics*, *134*(2), 85-86.

Pompanon, F., Bonin, A., Bellemain, E., & Taberlet, P. (2005). Genotyping errors: causes, consequences and solutions. *Nature reviews. Genetics*, *6*(11), 847.

Pop M, Salzberg SL (2008) Bioinformatics challenges of new sequencing technology. *Trends in Genetics* **24**(3): 142-149.

Potter SC, Clarke L, Curwen V, Keenan S, Mongin E, Searle SM et al (2004). The Ensembl analysis pipeline. *Genome research* **14**(5): 934-941.

R Core Team (2013). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. [http://www.R-project.org/]

Schuster SC (2008). Next-generation sequencing transforms today's biology. *Nature methods* *5*(1): 16-18.

Stephens ZD, Lee SY, Faghri F, Campbell RH, Zhai C, Efron MJ et al. (2015) Big data: astronomical or genomical?. *PLoS Biology* **13**(7): e1002195.

Strozzi F, Bonnal RJP (2017) Pipengine: an ultra light YAML-based pipeline execution engine. *The Journal of Open Source Software* **12**:16.

Sikorska K, Lesaffre E, Groenen PF, Eilers PH (2013). GWAS on your notebook: fast semi-parallel linear and logistic regression for genome-wide association studies. *BMC Bioinformatics* **14**(1): 166.

Szymczak S, Biernacka JM, Cordell HJ, González-Recio O, König IR, Zhang H, Sun YV (2009) Machine learning in genome-wide association studies. *Genetic epidemiology* **33**(S1): S51-57.

Taylor RC. (2010) An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics. *BMC Bioinformatics* **11**(Suppl 12):S1.

Tempelman, R. J. "The frontier spirit and reproducible research in animal breeding." *Journal of Animal Breeding and Genetics* 133, no. 6 (2016): 441-442.

VanRaden PM, O'Connell JR, Wiggans GR, Weigel KA (2011) Genomic evaluations with many more genotypes. *Genetics Selection Evolution* **43**(1):1.

Van Rossum G, Drake Jr FL (1995) *Python reference manual*. Amsterdam: Centrum voor Wiskunde en Informatica.

Vouzis PD, Sahinidis NV (2011). GPU-BLAST: using graphics processors to accelerate protein sequence alignment. *Bioinformatics* **27**(2): 182–188.

Tables

Table 1: Examples of Big Data from ’omics technologies

Examples of Big Data from ’omics technologies		
Category	**Raw data**	**Size**
Whole-genome sequences (WGS)	sequence reads	~ 5 GB for a genome ~ 3 Gbps long at ~ 10x coverage
Transcriptome Sequence Analysis (TSA)	sequence reads	several GB depending on coverage (< WGS)
Bisulphite sequencing	sequence reads	several GB (≤ TSA)
SNP array	genotypes	few kB for sample → usually several ples → MB/GB
5 GB: giga-bytes; 5 MB: mega-bytes; 5 kB: kilo-bytes; Gbps: giga-base-pairs.		

Table 2: Publicly available resources

Resource	Name	access	type
Forum	SEQanswers	http://seqanswers.com/	Sequencing, Bioinformatics
	Biostars	https://www.biostars.org/	Bioinformatics, Biological Data Analysis
	Stack Overflow	http://stackoverflow.com/	Informatics
	Stack Exchange	http://stackexchange.com/	Informatics
Software	Sourceforge	http://sourceforge.net/	Repository
	Github	https://github.com/	Repository
	Google Code	https://code.google.com/	Repository
	sqLite	https://www.sqlite.org	Database software
	YAML	http://yaml.org/	Data serialization standard
Database	Pubmed	http://www.ncbi.nlm.nih.gov/pubmed	Literature
	Scopus	http://www.scopus.com/	Literature
	Genbank	http://www.ncbi.nlm.nih.gov/genbank/	Data
	Ensembl	http://www.ensembl.org/index.html	Data

	Short Read Archive	http://www.ncbi.nlm.nih.gov/sra http://www.ebi.ac.uk/ena	Data
	Dryad	http://datadryad.org/	Data
	USGC Genome Browser	https://genome.ucsc.edu/	Data
Large Scale Projects	1000 genomes	http://www.1000genomes.org/	Human genomes
	1000 bull genomes project	http://www.1000bullgenomes.com/	Cattle genomes
	NextGen Consortium	http://nextgen.epfl.ch/	Mouflon, Sheep, Bezoar, Goat, Cattle
	The 3000 rice genomes project	http://gigadb.org/dataset/200001	Rice
	1001genomes	http://1001genomes.org/	Arabidopsis

Figures

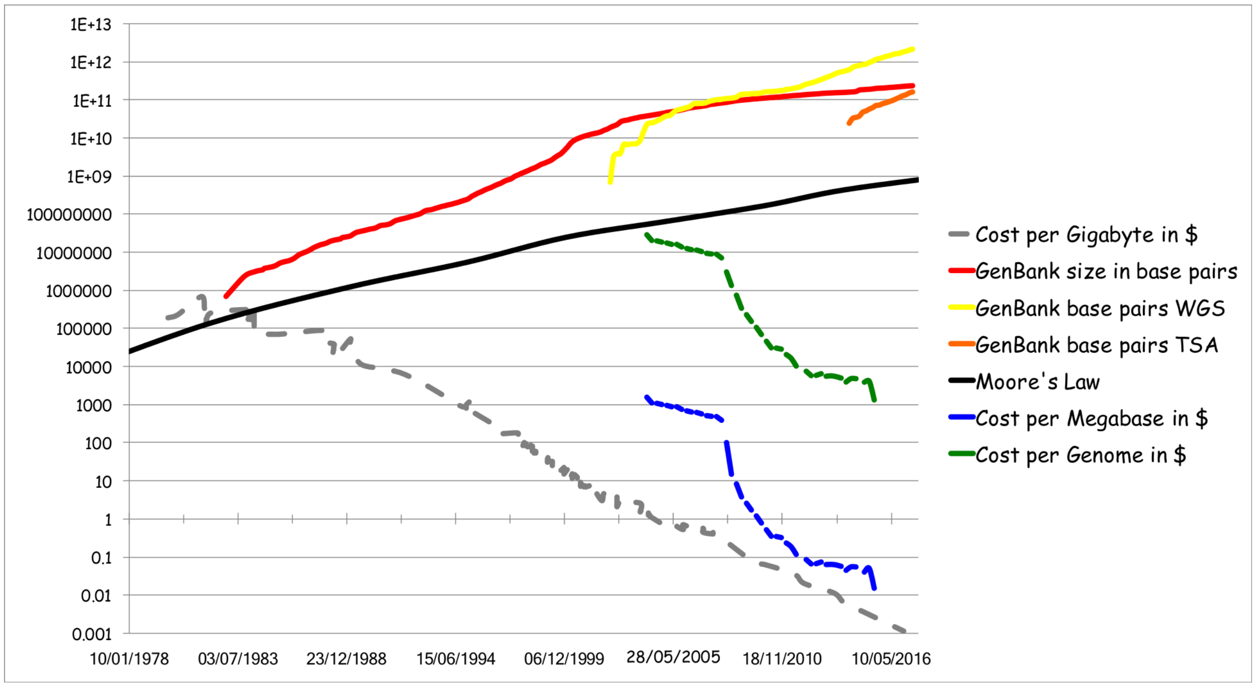
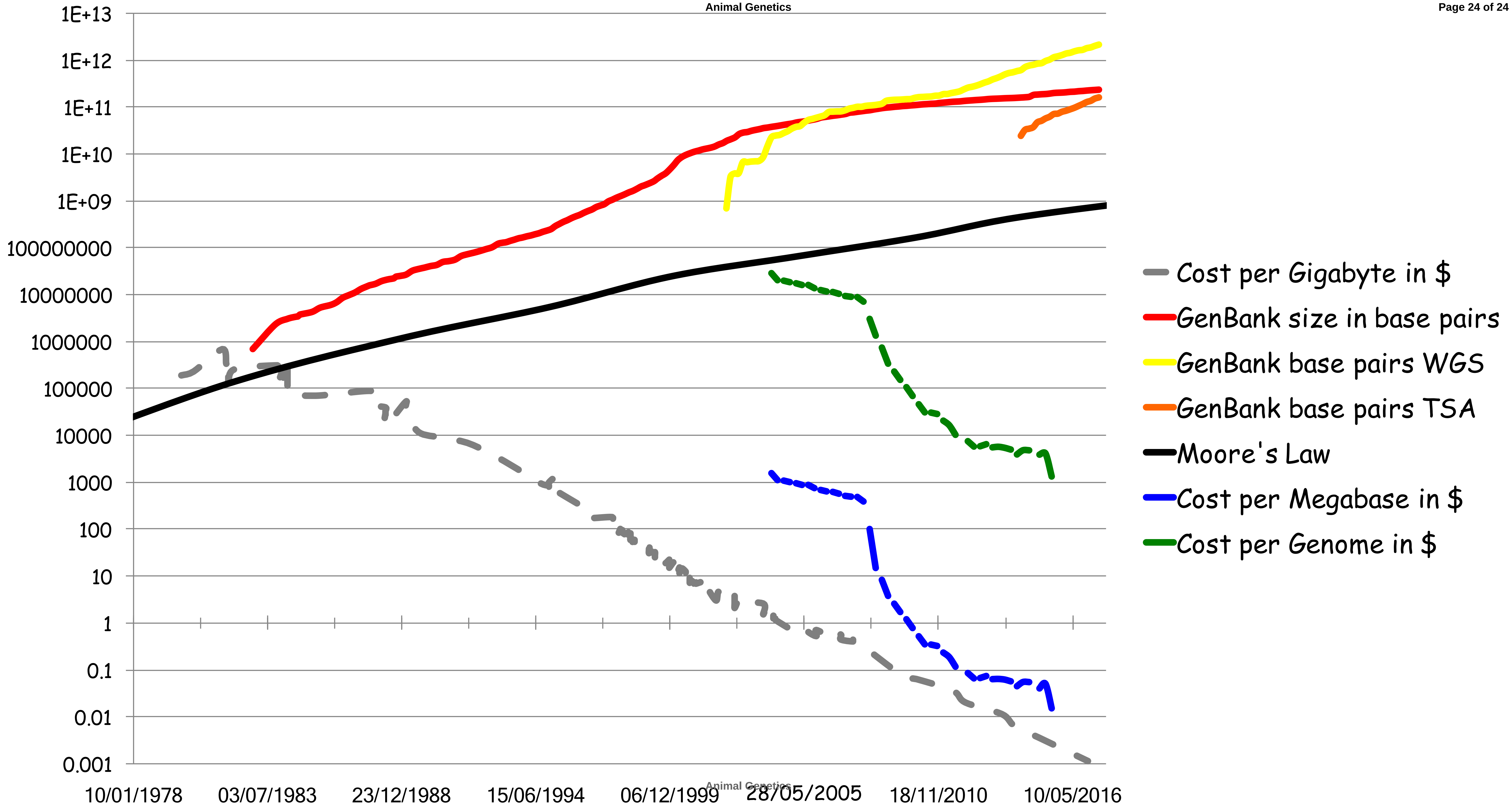


Figure 1: Trends in costs and data production over time. Cost per giga-byte (gray line), per genome (green line), per mega-base (blue line). Base-pairs from GenBank (red line), from whole-genome sequences (WGS, yellow line) and from transcriptome sequence analysis (TSA,

orange line); Moore’s law (black line). The y-axis holds for all units (dollars, base-pairs, n. of transistors). WGS and TSA data are not distributed in conjunction with GenBank releases. Data from ftp://ftp.ncbi.nih.gov/genbank/gbrel.txt, https://www.genome.gov/sequencingcosts/


1E+13
1E+12
1E+11
1E+10
1E+09
100000000
10000000
1000000
100000
10000
1000
100
10
1
0.1
0.01
0.001
10/01/1978
03/07/1983
23/12/1988
15/06/1994
06/12/1999
28/05/2005
18/11/2010
10/05/2016
Cost per Gigabyte in $
GenBank size in base pairs
GenBank base pairs WGS
GenBank base pairs TSA
Moore's Law
Cost per Megabase in $
Cost per Genome in $