Part I

Preliminaries

1 Molecular biology and high-throughput sequencing

In this chapter we give a minimalistic, combinatorial introduction to molecular biology, omitting the description of most biochemical processes and focusing on inputs and outputs, abstracted as mathematical objects. Interested readers might find it useful to complement our abstract exposition with a molecular biology textbook, to understand the chemical foundations of what we describe.

1.1 DNA, RNA, proteins

Life consists of fundamental units, called *cells*, that interact to form the complex, emergent behavior of a colony or of a multicellular organism. Different parts of a multicellular organism, like organs and tissues, consist of specialized cells that behave and interact in different ways. For example, muscle cells have a fundamentally different function and structure from brain cells. To understand such differences, one should look inside a cell.

A cell is essentially a *metabolic network* consisting of a mixture of molecules that interact by means of chemical reactions, with new "output" molecules constantly produced from "input" molecules. A set of reactions that are connected by their input and output products is called *pathway*. Each reaction is facilitated or hindered by a specific set of molecules, called *enzymes*, whose regulation affects the status of the entire network inside a cell. Enzymes are a specific class of *proteins*, linear chains of smaller building blocks (called *amino acids*) which can fold into complex three-dimensional structures. Most proteins inside a human cell consist of amino acids taken from a set of 20 different types, each with peculiar structural and chemical properties.

The environment in which a cell is located provides input molecules to its metabolic network, which can be interpreted as signals to activate specific pathways. However, the cell can also regulate its own behavior using a set of instructions *stored inside itself*, which essentially specify how to assemble new proteins, and at what rate. In extreme simplification, this "program" is immutable, but it is executed dynamically, since the current concentration of proteins inside a cell affects which instructions are executed at the next time point. The conceptual network that specifies which proteins affect which instruction is called the *regulation network*. To understand its logic, we need to look at the instruction set more closely.

In surprising analogy to computer science, the material that stores the instruction set consists of a collection of *sequential*, chain-like molecules, called *deoxyribonucleic acid (DNA)*. Every such molecule consists of the linear concatenation of smaller building blocks, called *nucleotides*. There are just four different types of nucleotides in a DNA molecule, called *bases*: adenine, cytosine, guanine, and thymine, denoted by `A`, `C`, `G`, and `T`, respectively. Every such sequence of nucleotides is called a *chromosome*, and the whole set of DNA sequences in a cell is called its *genome*. A DNA molecule is typically *double-stranded*, meaning that it consist of two chains (called *strands*), bound around each other to form a double helix. The sequence of nucleotides in the two strands is identical, up to a systematic replacement of every `A` with a `T` (and vice versa), and of every `C` with a `G` (and vice versa): such pairs of bases are called *complementary*, and the attraction of complementary base pairs, among other forces, holds the two strands together. The redundancy that derives from having two identical sequences for each chromosome up to replacement is an ingenious mechanism that allows one to *chemically copy* a chromosome: in extreme simplification, copying happens by separating the two strands, and by letting the bases of each isolated strand spontaneously attract complementary, unbound bases that float inside the cell. The ability to copy the instruction set is one of the key properties of life. Owing to the chemical mechanism by which consecutive nucleotides in the same strand are connected to each other, each strand is assigned a direction. Such directions are symmetrical, in the sense that if a strand is read from left to right, its complementary strand is read from right to left.

Some subsequences of a chromosome *encode* proteins, in the sense that there is a molecular mechanism that *reads* such subsequences and that interprets them as sequential instructions to concatenate amino acids into proteins. Such program-like subsequences are called *genes*. Specifically, a gene consists of a set of *exons*, contiguous, typically short subsequences of DNA that are separated by large subsequences called *introns* (see Figure 1.1). Exons and introns are ordered according to the direction of the strand to which they belong. A *transcript* is an ordered subset of the exons of a gene. The same gene can have multiple transcripts.

Transcription is the first step in the production of a protein from a gene. In extreme simplification, the two strands of a chromosome are first separated from each other around the area occupied by the gene, and then a *chain of ribonucleic acid (RNA)* is produced from one of the strands by a molecular machine that reads the gene *sequentially* and "prints" a complementary copy. RNA is like DNA, but thymine (`T`) is replaced by *uracil* (`U`), which becomes the complement of adenine (`A`). This first RNA copy of a gene is called the *primary transcript*, or *pre-mRNA*, where the letter "m" underlines the fact that it encodes a message to be converted into proteins, and the prefix "pre" implies that it is a precursor that still needs to be processed.

Further processing consists in extracting subsets of exons from the pre-mRNA, and in combining them into one or more predefined transcripts, discarding introns altogether. This biochemical process is called *splicing*, and the creation of multiple transcripts from the same gene is called *alternative splicing* (see Figure 1.1). Alternative splicing is heavily exploited by complex organisms to encode a large number of alternative proteins from a small set of genes. The beginning and the end

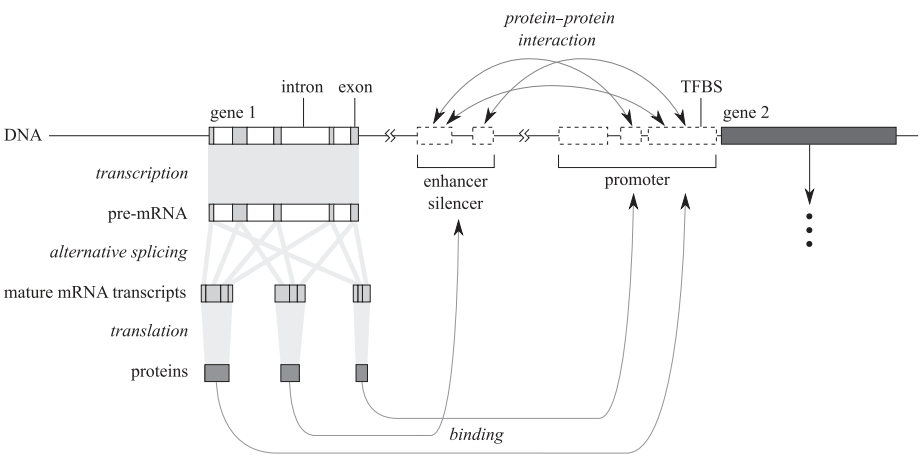

Figure 1.1 A schematic illustration of the central dogma. Gene 1 has three alternatively spliced transcripts. The relative expression of such transcripts affects the regulatory modules of gene 2, and eventually its expression. Definitions are given in Section 1.1.

of introns are typically marked by special subsequences that signal the positions at which introns must be cut out of the primary transcript (for example dinucleotide `GT` at the beginning, and dinucleotide `AG` at the end of an intron). The output of splicing is called *mature messenger RNA (mRNA)*. Introns are mostly absent from simple forms of life, like bacteria, and the entire process of transcription is much simpler in such cases.

The final step to convert mRNA into protein is called *translation*: in extreme simplification, a complex molecular machine reads a mature mRNA transcript *sequentially*, from left to right, three bases at a time, converting every such triplet (called a *codon*) into a specific amino acid determined by a fixed translation table, and chaining such amino acids together to form a protein. This universal table, shared by most organisms, is called *the genetic code*. There are $4^3 = 64$ distinct codons, so different codons are converted to the same amino acid: this redundancy is typically exploited by organisms to tolerate deleterious mutations in genes, or to tune the efficiency of translation. Specific *start* and *stop codons* signal the beginning and the end of the translation process. In addition to signalling, the start codon codes for a specific amino acid. Table 1.1 shows the genetic code used by the human genome.

This unidirectional flow of information from DNA to mRNA to protein, in which sequential information is neither transferred back from protein to RNA or DNA, nor from RNA to DNA, is called the *central dogma* of molecular biology, and it is summarized in Figure 1.1. The actual biological process is more complex than this dogma, and includes exceptions, but this abstraction is accurate enough to help understand the topics we will cover in the following chapters.

As mentioned earlier, proteins can work as enzymes to facilitate or inhibit specific chemical reactions in the metabolic network of a cell. Proteins also make up most of the structural parts of the cell, contributing to its shape, surface, and mobility. Finally,

Table 1.1. The genetic code used by the human genome. Amino acids (AA) are abbreviated by letters. Symbols ▷ and ◁ denote the start and stop signals, respectively.

AA	Codons	AA	Codons
A	GCT, GCC, GCA, GCG	N	AAT, AAC
C	TGT, TGC	P	CCT, CCC, CCA, CCG
D	GAT, GAC	Q	CAA, CAG
E	GAA, GAG	R	CGT, CGC, CGA, CGG, AGA, AGG
F	TTT, TTC	S	TCT, TCC, TCA, TCG, AGT, AGC
G	GGT, GGC, GGA, GGG	T	ACT, ACC, ACA, ACG
H	CAT, CAC	V	GTT, GTC, GTA, GTG
I	ATT, ATC, ATA	W	TGG
K	AAA, AAG	Y	TAT, TAC
L	TTA, TTG, CTT, CTC, CTA, CTG	◁	TAA, TGA, TAG
M, ▷	ATG		

specific proteins, called *transcription factors*, control the rate of the transcription process itself: indeed, the rate at which a gene is transcribed, called the *expression level* of the gene, is controlled by the binding of transcription factors to specific *regulatory regions* associated with the gene. Such proteins form molecular complexes called *enhancer* or *silencer modules*, which respectively facilitate or inhibit the formation of the molecular machine required to start transcription. In extreme simplification, this is the way in which the concentration of proteins in the cell at a given time point affects which instructions will be executed at the next time point.

The subsequences of DNA to which transcription factors bind are called *transcription factor binding sites* (TFBS), and they can be located both in a *promoter area* close to the beginning of a gene and at a large sequential distance before (*upstream* of) the beginning of a gene (see Figure 1.1). The double helix of DNA that forms a chromosome folds on itself at multiple scales, thus large distances along the sequence can translate into small distances in three-dimensional space, putting upstream transcription factors in the vicinity of a gene. Alternative splicing is itself regulated by specific proteins that bind to primary transcripts.

1.2 Genetic variations

The genome of a species is copied from individual to individual across multiple generations of a population, accumulating, like a chain letter, minimal differences at every step. The genomes of different individuals of the same species have typically the same number of chromosomes and most of the same bases in each chromosome: we can thus talk about the *consensus* or the *reference genome* of a species.

Variations from the consensus make every individual unique: most such variations are *single-nucleotide polymorphisms (SNPs)*, or equivalently variations of a single base. Insertions and deletions (in short, *indels*) are also common, with short indels

being more frequent than large indels. Large-scale *structural variants*, like the *inversion* of entire subsequences of a chromosome, the *translocation* of a subsequence to a different location or to a completely different chromosome, the *fusion* of two chromosomes that get concatenated to one another, or the *fission* of a chromosome into two or more fragments, have the potential to induce *speciation events* that create new species, but they are typically *deleterious* and cause lethal anomalies.

How the genetic information transfers from one generation to the next differs according to the strategy adopted by a species to reproduce. *Sexual reproduction* is common in *eukaryotes* – complex organisms whose cells use internal membranes to store genetic material and other special-purpose structures. In sexual reproduction, every cell of an individual is endowed with two copies of each chromosome, one inherited from a "mother" and the other from a "father". Such copies are called *homologous chromosomes* or *haplotypes*: a human individual has 23 pairs of homologous chromosomes, one of which determines its sex. A cell with two copies of each chromosome is called *diploid*. A specific set of cells inside an individual, called *germ cells*, gives rise to another set of specialized cells, called *gametes*, that contain only one copy of each chromosome and are used by an individual to combine its chromosomes with those of another individual. Thus, mutations that occur in germ cells, called *germline mutations*, can be inherited by the offspring of an individual that reproduces sexually. *Somatic mutations* are instead those variations that do not affect germ cells, and thus cannot be inherited by the offspring of an individual. Somatic mutations that occur early in the development of an individual can, however, spread to a large fraction of its cells.

Before producing gametes, a germ cell randomly *recombines* the two homologous copies of every chromosome, so that the single copy given to each gamete consists of a random sequence of consecutive fragments, taken alternatively from one haplotype and from the other at corresponding positions, as simplified below:

```
ACGACGAtcggagcgatgAGTCGAGctagct
accacgaTCCCAGAGATGtgtccagCTACCT
```

In this example, lower-case letters denote one haplotype, and upper-case letters denote the other haplotype. Variants that occur in a single haplotype are called *heterozygous*, while variants that occur in both haplotypes are called *homozygous*. Every possible variant that can be found at a specific position of a chromosome is called an *allele*.

1.3 High-throughput sequencing

Sequencing is the process of deriving the sequence of bases that compose a DNA fragment. It is typically hard to sequence very long fragments, like entire chromosomes, from beginning to end. However, it is relatively easy to cut a long fragment into short pieces, create multiple copies of each piece (a process known as *amplification*), and sequence all the resulting pieces: this process is called *shotgun sequencing*, and the sequence of every piece is called a *read*. A read can originate from either of the

two strands of a chromosome, and in the case of diploid cells from either of the two haplotypes. Sequencing loses the information about the position of a piece with respect to the original DNA fragment, therefore inferring the sequence of a chromosome from a corresponding set of reads is a nontrivial combinatorial puzzle, called *fragment assembly*, that we will solve in Chapter 13 by finding overlaps between pairs of reads. The term *high-throughput sequencing* refers to the fact that it is becoming increasingly cheaper to get a very large number of reads from a genome, or equivalently to cover every base of a genome with a large number of reads.

The technology to cut, amplify, and sequence DNA is under continuous development: rather than attempting a detailed description of the state of the art, we focus here on some fundamental concepts that most technologies have in common. By *short reads* we mean sequences whose length ranges from tens to hundreds of nucleotides: such reads typically contain just a few *sequencing errors*, most of which are single-base *substitutions* in which a base is mistaken for another base. By *long reads* we mean sequences that contain thousands of nucleotides: such reads are typically affected by a higher error rate.

For many species, accurate approximations of entire consensus genomes have been compiled by so-called *de novo sequencing* projects. Not surprisingly, the consensus human genome is quite accurate, and almost all of its roughly three gigabases are known. Common variants in the human population have also been mapped. As we will see in Chapter 10, it is relatively easy to detect SNP variants using a consensus genome and a set of reads from a specific individual, called a *donor*: indeed, it suffices to map every read to the position in the consensus genome that is "most similar" to it, and to study the resulting *pileup* of reads that cover a given position to detect whether the individual has a variation at that position. Collecting reads from a donor that belongs to an already sequenced species is called *whole genome resequencing*. In Chapter 10 we will study how reads can be mapped to their best matching locations using *sequence alignment* algorithms.

Figure 1.2 summarizes some key applications of high-throughput sequencing, in addition to the already mentioned *de novo* sequencing and whole genome resequencing. *RNA sequencing* allows one to sequence the complementary DNA *of RNA transcripts*: in this case, a read should be aligned to the reference genome allowing for one or more introns to split the read (see Section 16.1). In *targeted resequencing*, arbitrary areas of a genome are selected using specific marker sequences, called *primers*; such regions are then isolated, amplified, and sequenced, and the task is to detect their position in the reference genome using sequence alignment. We will mention an algorithm to compute primers in Section 11.3.3. In *chromatin immunoprecipitation sequencing* (often abbreviated to *ChIP-seq*) areas of the genome that contain the binding site of a specific set of transcription factors (or other DNA-binding proteins) are similarly isolated, amplified, and sequenced. Along the same lines, *bisulfite sequencing* can isolate and sequence areas of the genome with *methylation*. Methylation is a process in which cytosines (C) acquire a specific chemical compound, called a *methyl group*, and it is an example of an *epigenetic change*, or equivalently of a biochemical change that affects transcription *without altering the sequence of the genome*: indeed, methylation in promoter regions is known to affect the transcription rate of a gene. The set of

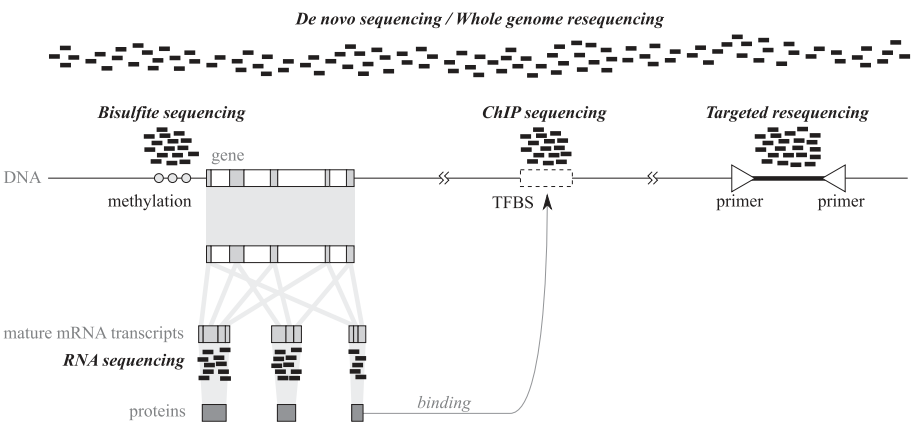

Figure 1.2 A schematic summary of high-throughput sequencing applications. Details are described in Section 1.3.

all epigenetic changes of a genome, some of which are inheritable, is called the *epigenome*.

Finally, when a biological sample contains more than one species, the set of all the genetic material present in the sample is called the *metagenome*. As we will see in Chapter 17, high-throughput sequencing can be used in this case to estimate the abundance of already sequenced species, and to reconstruct parts of the chromosomes of unknown species.

Exercises

1.1 Write a program that lists all the DNA sequences that encode a given protein sequence.

1.2 In a given organism some codons are used more frequently than others to encode the same amino acid. Given the observed frequency of every codon in a species, normalize it into probabilities and write a program that, given a protein sequence, samples a random DNA sequence that encodes that protein under such codon usage probabilities.

1.3 In a given organism, some *codon pairs* occur less frequently than others.

1. Given the set of all exons of an organism, write a program that computes the ratio $z(XY)$ between the observed and the *expected* number of occurrences of every pair of consecutive codons XY. Note that the expected number of occurrences of pair XY depends on the frequency with which X and Y are used to encode their corresponding amino acids.
2. Given a DNA sequence S that encodes a protein P, let $f(S)$ be the average of $z(XY)$ over all consecutive pairs of codons XY in S. Write a program that computes, if it exists, a permutation S' of the codons of S, such that S' still encodes P, but $f(S') < f(S)$. Such an *artificially attenuated* version of S has been shown to decrease the rate of protein translation: for more details, see Coleman et al. (2008).

2 Algorithm design

Among the general algorithm design techniques, *dynamic programming* is the one most heavily used in this book. Since great parts of Chapters 4 and 6 are devoted to introducing this topic in a unified manner, we shall not introduce this technique here. However, a taste of this technique is already provided by the solution to Exercise 2.2 of this chapter, which asks for a simple one-dimensional instance.

We will now cover some basic primitives that are later implicitly assumed to be known.

2.1 Complexity analysis

The algorithms described in this book are typically analyzed for their *worst-case* running time and space complexity: complexity is expressed as a function of the input parameters on the worst possible input. For example, if the input is a string of length n from an alphabet of size σ, a *linear time* algorithm works in $O(n)$ time, where $O(\cdot)$ is the familiar big-O notation which hides constants. This means that the running time is upper bounded by cn elementary operations, where c is some constant.

We consider the alphabet size not to be a constant, that is, we use expressions of the form $O(n\sigma)$ and $O(n \log \sigma)$, which are not linear complexity bounds. For the space requirements, we frequently use notations such as $n \log \sigma(1 + o(1)) = n \log \sigma + o(n \log \sigma)$. Here $o(\cdot)$ denotes a function that grows asymptotically strictly slower than its argument. For example, $O(n \log \sigma / \log \log n)$ can be simplified to $o(n \log \sigma)$. Algorithms have an input and an output: by *working space* we mean the extra space required by an algorithm in addition to its input and output. Most of our algorithms are for processing DNA, where $\sigma = 4$, and it would seem that one could omit such a small constant in this context. However, we will later see a number of applications of these algorithms that are not on DNA, but on some derived string over an alphabet that can be as large as the input sequence length.

We often assume the alphabet of a string of length n to be $\Sigma = [1..\sigma]$, where $\sigma \leq n$. Observe that if a string is originally from an *ordered alphabet* $\Sigma \subseteq [1..u]$, with $\sigma = |\Sigma|$, it is easy to map the alphabet and the character occurrences in the string into alphabet $[1..\sigma]$ in $O(n \log \sigma)$ time using, for example, a *binary search tree*: see Section 3.1. Such mapping is not possible with an *unordered alphabet* Σ that supports only the operation $a = b$? for characters $a, b \in \Sigma$: comparison $a < b$? is not supported.

Yet still, the running time of some algorithms will be $O(dn^c)$, where c is a constant and d is some integer given in the input that can take arbitrarily large values, independently of the input size n. In this case we say that we have a *pseudo-polynomial* algorithm. More generally, a few algorithms will have running time $O(f(k)n^c)$, where c is a constant and $f(k)$ is an arbitrary function that does not depend on n but depends only on some other – typically smaller – parameter k of the input. In this case, we say that we have a *fixed-parameter tractable* algorithm, since, if k is small and $f(k)$ is sufficiently slow-growing, then this algorithm is efficient, thus the problem is *tractable*.

In addition to the notations $o(\cdot)$ and $O(\cdot)$, we will also use the notations $\Theta(\cdot)$, $\omega(\cdot)$, and $\Omega(\cdot)$ defined as follows. By $\Theta(\cdot)$ we denote a function that grows asymptotically as fast as its argument (up to a constant factor). Whenever we write $f(x) = \Theta(g(x))$, we mean that $f(x) = O(g(x))$ and $g(x) = O(f(x))$. The notions $\omega(\cdot)$ and $\Omega(\cdot)$ are the inverses of $o(\cdot)$ and $O(\cdot)$, respectively. Whenever we write $f(x) = \omega(g(x))$, we mean that $g(x) = o(f(x))$. Whenever we write $f(x) = \Omega(g(x))$, we mean that $g(x) = O(f(x))$.

None of the algorithms in this book will require any complex analysis technique in order to derive their time and space requirements. We mainly exploit a series of combinatorial properties, which we sometimes combine with a technique called *amortized analysis*. In some cases it is hard to bound the running time of a single step of the algorithm, but, by finer counting arguments, the total work can be shown to correspond to some other conceptual steps of the algorithm, whose total amount can be bounded. Stated differently, the time "costs" of the algorithm can all be "charged" to some other bounded resource, and hence "amortized". This technique deserves a concrete example, and one is given in Example 2.1.

Example 2.1: Amortized analysis – Knuth–Morris–Pratt

The *exact string matching* problem is that of locating the occurrences of a *pattern* string $P = p_1 p_2 \cdots p_m$ inside a *text* string $T = t_1 t_2 \cdots t_n$. The Morris–Pratt (MP) algorithm solves this problem in optimal linear time, and works as follows. Assume first that we already have available on P the values of a *failure function*, $\texttt{fail}(i)$, defined as the length of the longest prefix of $p_1 p_2 \cdots p_i$ that is also a suffix of $p_1 p_2 \cdots p_i$. That is, $\texttt{fail}(i) = i'$, where $i' \in [0..i-1]$ is the largest such that $p_1 p_2 \cdots p_{i'} = p_{i-i'+1} p_{i-i'+2} \cdots p_i$. The text T can be scanned as follows. Compare p_1 to t_1, p_2 to t_2, and so on, until $p_{i+1} \neq t_{j+1}$, where j is an index in T and equals i throughout the first iteration. Apply $i = \texttt{fail}(i)$ recursively until finding $p_{i+1} = t_{j+1}$, or $i = 0$; denote this operation by $\texttt{fail}^*(i, t_{j+1})$. Then, continue the comparisons, by again incrementing i and j, and again resetting $i = \texttt{fail}^*(\cdot,\cdot)$ in case of mismatches. An occurrence of the pattern can be reported when $i = |P| + 1$.

To analyze the running time, observe first that each character t_j of T is checked for equality with some character of P exactly once. Thus, we need only estimate the time needed for the recursive calls $\texttt{fail}^*(i, t_{j+1})$. However, without a deeper analysis, we can only conclude that each recursive call takes $O(m)$ time before one can increment j, which would give an $O(mn)$ bound for the time complexity of the algorithm. In other words, it is *not* enough to consider the worst-case running time of each step independently, and then just sum the results. However, we can show that