Bioinformatics—An Introduction for Computer Scientists

JACQUES COHEN

Brandeis University

Abstract. The article aims to introduce computer scientists to the new field of bioinformatics. This area has arisen from the needs of biologists to utilize and help interpret the vast amounts of data that are constantly being gathered in genomic research—and its more recent counterparts, proteomics and functional genomics. The ultimate goal of bioinformatics is to develop in silico models that will complement in vitro and in vivo biological experiments. The article provides a bird's eye view of the basic concepts in molecular cell biology, outlines the nature of the existing data, and describes the kind of computer algorithms and techniques that are necessary to understand cell behavior. The underlying motivation for many of the bioinformatics approaches is the evolution of organisms and the complexity of working with incomplete and noisy data. The topics covered include: descriptions of the current software especially developed for biologists, computer and mathematical cell models, and areas of computer science that play an important role in bioinformatics.

Categories and Subject Descriptors: A.1 [**Introductory and Survey**]; F.1.1 [**Computation by Abstract Devices**]: Models of Computation—*Automata (e.g., finite, push-down, resource-bounded)*; F.4.2 [**Mathematical Logic and Formal Languages**]: Grammars and Other Rewriting Systems; G.2.0 [**Discrete Mathematics**]: General; G.3 [**Probability and Statistics**]; H.3.0 [**Information Storage and Retrieval**]: General; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search; I.5.3 [**Pattern Recongnition**]: Clustering; I.5.4 [**Pattern Recongnition**]: Applications—*Text processing*; I.6.8 [**Simulation and Modeling**]: Types of Simulation—*Continuous; discrete event*; I.7.0 [**Document and Text Processing**]: General; J.3 [**Life and Medical Sciences**]: *Biology and genetics*

General Terms: Algorithms, Languages, Theory

Additional Key Words and Phrases: Molecular cell biology, computer, DNA, alignments, dynamic programming, parsing biological sequences, hidden-Markov-models, phylogenetic trees, RNA and protein structure, cell simulation and modeling, microarray

1. INTRODUCTION

It is undeniable that, among the sciences, biology played a key role in the twentieth century. That role is likely to acquire further importance in the years to come. In the wake of the work of Watson and Crick, [2003] and the sequencing of the human genome, far-reaching discoveries are constantly being made.

One of the central factors promoting the importance of biology is its relationship with medicine. Fundamental progress in medicine depends on elucidating some of

Author's address: Department of Computer Science, Brandeis University, Waltham, MA 02454; email: jc@cs.brandeis.edu.

the mysteries that occur in the biological sciences

Biology depended on chemistry to make major strides, and this led to the development of biochemistry. Similarly, the need to explain biological phenomena at the atomic level led to biophysics. The enormous amount of data gathered by biologists—and the need to interpret it—requires tools that are in the realm of computer science. Thus, bioinformatics.

Both chemistry and physics have benefited from the symbiotic work done with biologists. The collaborative work functions as a source of inspiration for novel pursuits in the original science. It seems certain that the same benefit will accrue to computer science—work with biologists will inspire computer scientists to make discoveries that improve their own discipline.

A common problem with the maturation of an interdisciplinary subject is that, inevitably, the forerunner disciplines call for differing perspectives. I see these differences in working with my biologist colleagues. Nevertheless, we are so interested in the success of our dialogue, that we make special efforts to understand each other's point of view. That willingness is critical for joint work, and this is particularly true in bioinformatics.

An area called *computational biology* preceded what is now called bioinformatics. Computational biologists also gathered their inspiration from biology and developed some very important algorithms that are now used by biologists. Computational biologists take justified pride in the formal aspects of their work. Those often involve proofs of algorithmic correctness, complexity estimates, and other themes that are central to theoretical computer science.

Nevertheless, the biologists' needs are so pressing and broad that many other aspects related to computer science have to be explored. For example, biologists need software that is reliable and can deal with huge amounts of data, as well as interfaces that facilitate the human-machine interactions.

I believe it is futile to argue the differences and scope of computational biology as compared to bioinformatics. Presently, the latter is more widely used among biologists than the former, even though there is no agreed definition for the two terms.

A distinctive aspect of bioinformatics is its widespread use of the Web. It could not be otherwise. The immense databases containing DNA sequences and 3D protein structures are available to almost any researcher. Furthermore, the community interested in bioinformatics has developed a myriad of application programs accessible through the Internet. Some of these programs (e.g., BLAST) have taken years of development and have been finely tuned. The vast numbers of daily visits to some of the NIH sites containing genomic databases are comparable to those of widely used search engines or active software downloading sites. This explains the great interest that bioinformaticians have in script languages such as Perl and Python that allow the automatic examination and gathering of information from websites.

With the above preface, we can put forward the objectives of this article and state the background material necessary for reading it. The article is both a tutorial and a survey. As its title indicates, it is oriented towards computer scientists.

Some biologists may argue that the proper way to learn bioinformatics is to have a good background in organic chemistry and biochemistry and to take a full course in molecular cell biology. I beg to disagree: In an interdisciplinary field like bioinformatics there must be several entry points and one of them is using the language that is familiar to computer scientists. This does not imply that one can skip the fundamental knowledge available in a cell and molecular biology text. It means that a computer scientist interested in learning what is presently being done in bioinformatics can save some precious time by reading the material in this article.

It was mentioned above that, in interdisciplinary areas like bioinformatics, the players often view topics from a different

perspective. This is not surprising since both biologists and computer scientists have gone through years of education in their respective fields. A related plausible reason for such differences is as follows:

In computer science, we favor generality and abstractions; our approach is often top-down, as if we were developing a program or writing a manual. In contrast, biologists often favor a bottom-up approach. This is understandable because the minutiae are so important and biologists are often involved with time-consuming experiments that may yield ambiguous results, which in turn have to be resolved by further tests. The work of synthesis eventually has to take place but, since in biology most of the rules have exceptions, biologists are wary of generalizations.

Naturally, in this article's presentation, I have used a top-down approach. To make the contents of the article clear and self-contained, certain details have had to be omitted. From a pragmatic point of view, articles like this one are useful in bridging disciplines, provided that the readers are aware of the complexities lying behind abstractions.

The article should be viewed as a tour of bioinformatics enabling the interested reader to search subsequently for deeper knowledge. One should expect to expend considerable effort gaining that knowledge because bioinformatics is inextricably tied to biology, and it takes time to learn the basic concepts of biology.

This work is directed to a mature computer scientist who wishes to learn more about the areas of bioinformatics and computational biology. The reader is expected to be at ease with the basic concepts of algorithms, complexity, language and automata theory, topics in artificial intelligence, and parallelism.

As to knowledge in biology, I hope the reader will be able to recall some of the rudiments of biology learned in secondary education or in an undergraduate course in the biological sciences. An appendix to this article reviews a minimal set of facts needed to understand the material in the subsequent sections. In addition, *The Cartoon Guide to Genetics* [Gonick and Wheelis 1991] is an informative and amusing text explaining the fundamentals of cell and molecular biology.

The reader should keep in mind the pervasive role of evolution in biology. It is evolution that allows us to infer the cell behavior of one species, say the human, from existing information about the cell behavior of other species like the mouse, the worm, the fruit fly, and even yeast.

In the past decades, biologists have gathered information about the cell characteristics of many species. With the help of evolutionary principles, that information can be extrapolated to other species. However, most available data is fragmented, incomplete, and noisy. So if one had to characterize bioinformatics in logical terms, it would be: *reasoning with incomplete information*. That includes providing ancillary tools allowing researchers to compare carefully the relationship between new data and data that has been validated by experiments. Since understanding the human cell is a primary concern in medicine, one usually wishes to infer human cell behavior from that of other species.

This article's contents aim to shed some light on the following questions:

—How can one describe the actors and processes that take place within a living cell?

—What can be determined or measured to infer cell behavior?

—What data is presently available for that purpose?

—What are the present major problems in bioinformatics and how are they being solved?

—What areas in computer science relate most to bioinformatics?

The next section offers some words of caution that should be kept in the reader's mind. The subsequent sections aim at answering the above questions. A final section provides information about how to proceed if one wishes to further explore this new discipline.

2. WORDS OF CAUTION

Naturally, it is impossible to condense the material presently available in many bioinformatics texts into a single survey and tutorial. Compromises had to be made that may displease purists who think that no shortcuts are possible to explain this new field. Nevertheless, the objective of this work will be fulfilled if it incites the reader to continue along the path opened by reading this précis.

There is a dichotomy between the various presentations available in bioinformatics articles and texts. At one extreme are those catering to algorithms, complexity, statistics, and probability. On the other are those that utilize tools to infer new biological knowledge from existing data. The material in this article should be helpful to initiate potential practitioners in both areas. It is always worthwhile to keep in mind that new algorithms become useful when they are developed into packages that are used by biologists.

It is also wise to recall some of the distinguishing aspects of biology to which computer scientists are not accustomed. David B. Searls, in a highly recommended article on *Grand challenges in computational biology* [Searls 1998], points out that, in biology:

—*There are no rules without exception;*
—*Every phenomenon has a nonlocal component;*
—*Every problem is intertwined with others.*

For example, for some time, it was thought that a gene was responsible for producing a single protein. Recent work in alternate splicing indicates that a gene may generate several proteins. This may explain why the number of genes in the human genome is smaller than that that had been anticipated earlier. Another example of the biology's fundamentally dynamic and empirical state is that it has been recently determined that gene-generated proteins may contain amino acids beyond the 20 that are normally used as constituents of those proteins.

The second item in Searls' list warns us that the existence of common local features cannot be generalized. For example, similar 3D substructures may originate from different sequences of amino acids. This implies that similarity at one level cannot be generalized to another.

Finally, the third item cautions us to consider biological problems as an aggregate and not to get lost in studying only individual components. For example, simple changes of nucleotides in DNA may result in entirely different protein structures and function. This implies that the study of genomes has to be tied to the study of the resulting proteins.

3. BRIEF DESCRIPTION OF A SINGLE CELL AT THE MOLECULAR LEVEL

In this section, we assume that the reader has a rudimentary level of knowledge in cell and molecular biology. (The appendix reviews some of that material.) The intent here is to show the importance three-dimensional structures have in understanding the behavior of a living cell. Cells in different organisms or within the same organism vary significantly in shape, size, and behavior. However, they all share common characteristics that are essential for life.

The cell is made up of molecular components, which can be viewed as 3D-structures of various *shapes*. These molecules can be quite large (like DNA molecules) or relatively small (like the proteins that make up the cell membrane). The membrane acts as a filter that controls the access of exterior elements and also allows certain molecules to exit the cell.

Biological molecules in isolation usually maintain their structure; however, they may also contain articulations that allow movements of their subparts (thus, the interest of nano-technology researchers in those molecules).

The intracellular components are made of various types of molecules. Some of them navigate randomly within the media inside the membrane. Other molecules are attracted to each other.

In a living cell, the molecules *interact* with each other. By interaction it is meant that two or more molecules are combined to form one or more new molecules, that is, new 3D-structures with new shapes. Alternatively, as a result of an interaction, a molecule may be disassembled into smaller fragments. An interaction may also reflect mutual influence among molecules. These interactions are due to attractions and repulsions that take place at the atomic level. An important type of interaction involves catalysis, that is, the presence of a molecule that facilitates the interaction. These facilitators are called *enzymes*.

Interactions amount to chemical reactions that change the energy level of the cell. A living cell has to maintain its orderly state and this takes energy, which is supplied by surrounding light and nutrients.

It can be said that biological interactions frequently occur because of the *shape and location* of the cell's constituent molecules. In other words, the proximity of components and the shape of components trigger interactions. Life exists only when the interactions can take place.

A cell grows because of the availability of external molecules (nutrients) that can penetrate the cell's membrane and participate in interactions with existing intracellular molecules. Some of those may also exit through the membrane. Thus, a cell is able to "digest" surrounding nutrients and produce other molecules that are able to exit through the cell's membrane. A *metabolic pathway* is a chain of molecular interactions involving enzymes. *Signaling pathways* are molecular interactions that enable communication through the cell's membrane. The notions of metabolic and signaling pathways will be useful in understanding gene regulation, a topic that will be covered later.

Cells, then, are capable of *growing* by absorbing outside nutrients. Copies of existing components are made by interactions among exterior and interior molecules. A living cell is thus capable of *reproduction*: this occurs when there are enough components in the original cell to produce a duplicate of the original cell, capable of acting independently.

So far, we have intuitively explained the concepts of growth, metabolism, and reproduction. These are some of the basic characteristics of living organisms. Other important characteristics of living cells are: motility, the capability of searching for nutrients, and eventually death.

Notice that we assumed the initial existence of a cell, and it is fair to ask the question: how could one possibly have engineered such a contraption? The answer lies in *evolution*. When the cell duplicates it may introduce slight (random) changes in the structure of its components. If those changes extend the life span of the cell they tend to be incorporated in future generations. It is still unclear what ingredients made up the primordial living cell that eventually generated all other cells by the process of evolution.

The above description is very general and abstract. To make it more detailed one has to introduce the differences between the various components of a cell. Let us differentiate between two types of cell molecules: DNA and proteins. DNA can be viewed as a template for producing additional (duplicate) DNA and also for producing proteins.

Protein production is carried out using cascading transformations. In bacterial cells (called *prokaryotes*), RNA is first generated from DNA and proteins are produced from RNA. In a more developed type of cells (*eukaryotes*), there is an additional intermediate transformation: pre-RNA is generated from DNA, RNA from pre-RNA, and proteins from RNA. Indeed, the present paragraph expresses what is known as the *central dogma* in molecular biology. (Graphical representations of these transformations are available in the site of the National Health Museum [http://www.accessexcellence.org/AB/GG/].)

Note that the above transformations are actually molecular interactions such as we had previously described. A transformation $A \rightarrow B$ means that the resulting

molecules of B are constructed anew using subcomponents that are "copies" of the existing molecules of A. (Notice the similarity with Lisp programs that use constructors like *cons* to carry out transformations of lists.)

The last two paragraphs implicitly assume the existence of *processors* capable of effecting the transformations. Indeed that is the case with RNA-polymerases, spliceosomes, and ribosomes. These are marvelous machineries that are made themselves of proteins and RNA, *which in turn are produced from DNA!* They demonstrate the omnipresence of loops in cell behavior.

One can summarize the molecular transformations that interest us using the notation:

$$\textbf{DNA} \xrightarrow[\textit{RNA-polymerase}]{} \textbf{pre-RNA} \xrightarrow[\textit{Spliceosome}]{} \textbf{RNA} \xrightarrow[\textit{Ribosome}]{} \textbf{Protein}$$

The arrows denote transformations and the entities below them indicate the processors responsible for carrying out the corresponding transformations. Some important remarks are in order:

(1) All the constituents in the above transformation are three-dimensional structures.

(2) It is more appropriate to consider a *gene* (a subpart of DNA) as the original material processed by RNA-polymerase.

(3) An arsenal of processors in the vicinity of the DNA molecule works on multiple genes simultaneously.

(4) The proteins generated by various genes are used as constituents making up the various processors.

(5) A generated protein may prevent (or accelerate) the production of other proteins. For example, a protein P_i may place itself at the origin of gene G_k and prevent P_k from being produced. It is said that P_i *represses* P_k. In other cases, the opposite occurs: one protein *activates* the production of another.

(6) It is known that a spliceosome is capable of generating different RNAs (alternate splicing) and therefore the old notion that a given gene produces one given protein no longer holds true. As a matter of fact, a gene may produce several different proteins, though the mechanism of this is still a subject of research.

(7) It is never repetitious to point out that in biology, most rules have exceptions [Searls 1998].

The term, *gene expression*, refers to the production of RNA by a given gene. Presumably, the amount of RNA generated by the various genes of an organism establishes an estimate of the corresponding protein levels.

An important datum that can be obtained by laboratory experiments is an estimate of the simultaneous RNA production of thousands of genes. Gene expressions vary depending on a given state of the cell (e.g., starvation or lack of light, abnormal behavior, etc.).

3.1. Analogy with Computer Science Programs

We now open a parenthesis to recall the relationship that exists between computer programs and data; that relationship has analogies that are applicable to understanding cell behavior. Not all biologists will agree with a metaphor equating DNA to a computer program. Nevertheless, I have found that metaphor useful in explaining DNA to computer scientists.

In the universal Turing Machine (TM) model of computing, one does not distinguish between program and data—they coexist in the machine's tape and it is the TM interpreter that is commanded to start computations at a given state examining a given element of the tape.

Let us introduce the notion of interpretation in our simplified description of a single biological cell. Both DNA and proteins are components of our model, but the interactions that take place between DNA and other components (existing proteins)

result in producing new proteins *each of which has a specific function needed for cell survival* (growth, metabolism, replication, and others).

The following statement is crucial to understanding the process of interpretation occurring within a cell. Let a gene G in the DNA component be responsible for producing a protein P. Interpreters I capable of processing any gene may well utilize P as one of its components. This implies that if P has not been assembled into the machinery of I no interpretation takes place.

Another instance in which P cannot be produced is the already mentioned fact that another protein P' may position itself at the beginning of gene G and (temporarily) prevent the transcription.

The interpreter in the biological case is either one that already exists in a given cell (prior to cell replication) or else it can be assembled from proteins and RNA generated by specific genes (e.g., ribosomal genes). In biology the interpreter can be viewed as a mechanical gadget that is made of moving parts that produce new components based on given templates (DNA or RNA). The construction of new components is made by subcomponents that happen to be in the vicinity. If they are not, interpretation cannot proceed.

One can imagine a similar situation when interpreting computer programs (although it is unlikely to occur in actual interpreters). Assume that the components of I are first generated on the fly and once I is assembled (as data), control is transferred to the execution of I (as a program).

The above situation can be simulated in actual computers by utilizing concurrent processes that comprise a multitude of interrupts to control program execution. This could be implemented using interpreters that first test that all the components have been assembled: execution proceeds only if that is the case; otherwise an interrupt takes place until the assembly is completed. Alternatively one can execute program parts as soon as they are produced and interrupt execution if a sequel has not yet been fully generated. In Section 7.5.1, we will describe one such model of gene interaction.

4. LABORATORY TOOLS FOR DETERMINING BIOLOGICAL DATA

We start with a warning that the explanations that follow are necessarily coarse. The goal of this section is to enable the reader to have some grasp of how biological information is gathered and of the degree of difficulty in obtaining it. This will be helpful in understanding the various types of data available and the programs needed to utilize and interpret that data.

Sequencers are machines capable of reading off a sequence of nucleotides in a strand of DNA in biological samples. The machines are linked to computers that display the DNA sequence being analyzed. The display also provides the degree of confidence in identifying each nucleotide. Present sequencers can produce over 300k base pairs per day at very reasonable costs. It is also instructive to remark that the inverse operation of sequencing can also be performed rather inexpensively: it is now common practice to order from biotech companies vials containing short sequences of nucleotides specified by a user.

A significant difficulty in obtaining an entire genome's DNA is the fact that the sequences gathered in a wet lab consist of relatively short random segments that have to be reassembled using computer programs; this is referred to as the *shotgun method* of sequencing. Since DNA material contains many repeated subsequences, performing the assemblage can be tricky. This is due to the fact that a fragment can be placed ambiguously in two or more positions of the genome being assembled. (DNA assembly will be revisited in Section 7.7.)

Recently, there has been a trend to attempt to identify proteins using mass spectroscopy. The technique involves determining genes and obtaining the corresponding proteins in purified form. These are cut into short sequences of amino acids (called *peptides*) whose molecular weights can be determined by a mass spectrograph. It is then computationally possible to infer the constituents of the peptides

yielding those molecular weights. By using existing genomic sequences, one can attempt to reassemble the desired sequence of amino acids.

The 3D structure of proteins is mainly determined by X-ray crystallography and by nuclear magnetic resonance (NMR). Both these experiments are time consuming and costly. In X-ray crystallography, one attempts to infer the 3D position of each of the protein's atoms from a projection obtained by passing X-rays through a crystallized sample of that protein. One of the major difficulties of the process is the obtaining of good crystals. X-ray experiments may require months and even years of laboratory work.

In the NMR technique, one obtains a number of matrices that express the fact that two atoms—that are not in the same backbone chain of the protein—are within a certain distance. One then deduces a 3D shape from those matrices. The NMR experiments are also costly. A great advantage is that they allow one to study mobile parts of proteins, a task which cannot be done using crystals.

The preceding paragraphs explain why DNA data is so much more abundant than 3D protein data.

Another type of valuable information obtainable through lab experiments is known as ESTs or *expressed sequence tags*. These are RNA chunks that can be gathered from a cell in minute quantities, but can easily be duplicated. Those chunks are very useful since they do not contain material that would be present in *introns* (see the Appendix for a definition). The availability of EST databases comprising many organisms allows bioinformaticians to infer the positions of introns and even deduce alternate splicing.

A powerful new tool available in biology is microarrays. They allow determining simultaneously the amount of mRNA production of thousands of genes. As mentioned earlier, this amount corresponds to *gene-expression*; it is presumed that the amount of RNA generated by the various genes of an organism establishes an estimate of the corresponding protein levels.

Microarray experiments require three phases. In the first phase one places thousands of different one-stranded chunks of RNA in minuscule wells on the surface of a small glass chip. (This task is not unlike that done by a jet printer using thousands of different colors and placing each of them in different spots of a surface.) The chunks correspond to the RNA known to have been generated by a given gene. The 2D coordinates of each of the wells are of course known. Some companies mass produce custom preloaded chips for cells of various organisms and sell them to biological labs.

The second phase consists of spreading—on the surface of the glass—genetic material (again one-stranded RNA) obtained by a cell experiment one wishes to perform. Those could be the RNAs produced by a diseased cell, or by a cell being subjected to starvation, high temperature, etc. The RNA already in the glass chip combines with the RNA produced by the cell one wishes to study. The degree of combined material obtained by complementing nucleotides is an indicator of how much RNA is being expressed by each one of the genes of the cell being studied.

The third phase consists of using a laser scanner connected to a computer. The apparatus measures the amount of combined material in each chip well and determines the degree of gene expression—a real number—for each of the genes originally placed on the chip. Microarray data is becoming available in huge amounts. A problem with this data is that it is noisy and its interpretation is difficult. Microarrays are becoming invaluable for biologists studying how genes interact with each other. This is crucial in understanding disease mechanisms.

The microarray approach has been extended to the study of protein expression. There exist chips whose wells contain molecules that can be bound to particular proteins.

Another recent development in experimental biology is the determination of protein interaction by what is called *two-hybrid* experiments. The goal of such

experiments is to construct huge Boolean matrices, whose rows and columns represent the proteins of a genome. If a protein interacts with another, the corresponding position in the matrix is set to true. Again, one has to deal with thousands of proteins (genes); the data of their interactions is invaluable in reconstructing metabolic and signaling pathways.

A final experimental tool described in this section is the availability of libraries of variants of a given organism, yeast being a notable example. Each variant corresponds to cells having a single one of its genes knocked out. (Of course researchers are only interested in living cells since certain genes are vital to life.) These libraries enable biologists to perform experiments (say, using microarray) and deduce information about cell behavior and fault tolerance.

A promising development in experimental biology is the use of RNA-*i* (the *i* denoting interference). It has been found that when chunks of the RNA of a given gene are inserted in the nucleus of a cell, they may prevent the production of that gene. This possibility is not dissimilar to that offered by libraries of knocked-out genes.

The above descriptions highlight the trend of molecular biology experiments being done by ordering components, or by having them analyzed by large biotech companies.

5. BIOLOGICAL DATA AVAILABLE

In a previous section, we mentioned that all the components of a living cell are 3D structures and that shape is crucial in understanding molecular interactions. A fundamental abstraction often done in biology is to replace the spatial 3D information specifying chemical bindings with a much simpler sequence of symbols: nucleotides or amino acids. In the case of DNA, we know that the helix is the underlying 3D structure.

Although it is much more convenient to deal with sequences of symbols than with complex 3D entities, the problem of shape determination remains a critical one in the case of RNA and proteins.

The previous section outlined the laboratory tools for gathering biological data. The vast majority of the existing information has been obtained through sequencing, and it is expressible by strings—that is, sequences of symbols. These sequences specify mostly nucleotides (genomic data) but there is also substantial information on sequences of amino acids.

Next in volume of available information are the results of microarray experiments. These can be viewed as very large usually dense matrices of real numbers. These matrices may have thousands of rows and columns. And that is also the case of the sparse Boolean matrices describing protein interactions.

The information about 3D structures of proteins pales in comparison to that available in sequence form. The protein database (PDB) is the repository for all known three-dimensional protein structures.

In a recent search, I found that there are now about 26 billion base pairs (bp) representing the various genomes available in the server of the National Center for Biotechnology Information (NCBI). Besides the human genome with about 3 billion bp, many other species have their complete genome available there. These include several bacteria (e.g., *E. Coli*) and higher organisms including yeast, worm, fruit fly, mouse, and plants (e.g., Arabidopsis).

The largest known gene in the NCBI server has about 20 million base pairs and the largest protein consists of about 34,000 amino acids. These figures give an idea of the lengths of the entities we have to deal with.

In contrast, the PDB has a catalogue of only 45,000 proteins specified by their 3D structure. These proteins originate from various organisms. The relatively meager protein data shows the enormous need of inferring protein shape from data available in the form of sequences. This is one of the major tasks facing biologists. But many others lie ahead.

The goal of understanding protein structure is only part of the task. Next we have to understand how proteins interact and

form the metabolic and signaling pathways in the cell.

There is information available about metabolic pathways in simple organisms, and parts of those pathways are known for human cells. The formidable task is to put all the available information together so that it can be used to understand better the functioning of the human cell. That pursuit is called *functional genomics*.

The term, *genomics*, is used to denote the study of various genomes as entities having similar contents. In the past few years other terms ending with the suffixes-*ome* or - *mics* have been popularized. That explains *proteomics* (the study of all the proteins of a genome), *transcriptome*, *metabolome*, and so forth.

6. PRESENT GOALS OF BIOINFORMATICS

The present role of bioinformatics is to aid biologists in gathering and processing genomic data to study protein function. Another important role is to aid researchers at pharmaceutical companies in making detailed studies of protein structures to facilitate drug design. Typical tasks done in bioinformatics include:

—*Inferring a protein's shape and function from a given a sequence of amino acids*,
—*Finding all the genes and proteins in a given genome,*
—*Determining sites in the protein structure where drug molecules can be attached.*

To perform these tasks, one usually has to investigate *homologous* sequences or proteins for which genes have been determined and structures are available. Homology between two sequences (or structures) suggests that they have a common ancestor. Since those ancestors may well be extinct, one hopes that similarity at the sequence or structural level is a good indicator of homology.

It is important to keep in mind that sequence similarity does not always imply similarity in structure, and vice-versa. As a matter of fact, it is known that two fairly dissimilar sequences of amino acids may fold into similar 3D structures.

Nevertheless, the search for similarity is central to bioinformatics. When given a sequence (nucleotides or amino acids) one usually performs a search of similarity with databases that comprise all available genomes and known proteins. Usually, the search yields many sequences with varying degrees of similarities. It is up to the user to select those that may well turn out to be homologous.

In the next section we describe the various computer science algorithms that are frequently used by bioinformaticians.

7. ALGORITHMS FREQUENTLY USED IN BIOINFORMATICS

We recall that a major role of bioinformatics is to help infer gene function from existing data. Since that data is varied, incomplete, noisy, and covers a variety of organisms, one has to constantly resort to the biological principles of evolution to filter out useful information.

Based on the availability of the data and goals described in Sections 4 to 6, we now present the various algorithms that lead to a better understanding of gene function. They can be summarized as follows:

(1) *Comparing Sequences.* Given the huge number of sequences available, there is an urgent need to develop algorithms capable of comparing large numbers of long sequences. These algorithms should allow the deletion, insertion, and replacements of symbols representing nucleotides or amino acids, for such transmutations occur in nature.

(2) *Constructing Evolutionary (Phylogenetic) Trees.* These trees are often constructed after comparing sequences belonging to different organisms. Trees group the sequences according to their degree of similarity. They serve as a guide to reasoning about how these sequences have been transformed through evolution. For example, they infer homology from similarity, and may rule out erroneous assumptions that contradict known evolutionary processes.

(3) *Detecting Patterns in Sequences.* There are certain parts of DNA and amino acid sequences that need to be detected. Two prime examples are the search for genes in DNA and the determining of subcomponents of a sequence of amino acids (secondary structure). There are several ways to perform these tasks. Many of them are based on machine learning and include probabilistic grammars, or neural networks.

(4) *Determining 3D Structures from Sequences.* The problems in bioinformatics that relate sequences to 3D structures are computationally difficult. The determination of RNA shape from sequences requires algorithms of cubic complexity. The inference of shapes of proteins from amino acid sequences remains an unsolved problem.

(5) *Inferring Cell Regulation.* The function of a gene or protein is best described by its role in a metabolic or signaling pathway. Genes interact with each other; proteins can also prevent or assist in the production of other proteins. The available approximate models of cell regulation can be either discrete or continuous. One usually distinguishes between cell *simulation* and *modeling*. The latter amounts to inferring the former from experimental data (say microarrays). This process is usually called *reverse engineering*.

(6) *Determining Protein Function and Metabolic Pathways.* This is one of the most challenging areas of bioinformatics and for which there is not considerable data readily available. The objective here is to interpret human annotations for protein function and also to develop databases representing graphs that can be queried for the existence of nodes (specifying reactions) and paths (specifying sequences of reactions).

(7) *Assembling DNA Fragments.* Fragments provided by sequencing machines are assembled using computers. The tricky part of that assemblage is that DNA has many repetitive regions and the same fragment may belong to different regions. The algorithms for assembling DNA are mostly used by large companies (like the former Celera).

(8) *Using Script Languages.* Many of the above applications are already available in websites. Their usage requires scripting that provides data for an application, receives it back, and then analyzes it.

The algorithms required to perform the above tasks are detailed in the following subsections. What differentiates bioinformatics problems from others is the huge size of the data and its (sometimes questionable) quality. That explains the need for approximate solutions.

It should be remarked that several of the problems in bioinformatics are constrained optimization problems. The solution to those problems is usually computationally expensive. One of the efficient known methods in optimization is dynamic programming. That explains why this technique is often used in bioinformatics. Other approaches like branch-and-bound are also used, but they are known to have higher complexity.

7.1. Comparing Sequences

From the biological point of view sequence comparison is motivated by the fact that all living organisms are related by evolution. That implies that the genes of species that are closer to each other should exhibit similarities at the DNA level; one hopes that those similarities also extend to gene function.

The following definitions are useful in understanding what is meant by the comparison of two or more sequences. An *alignment* is the process of lining up sequences to achieve a maximal level of identity. That level expresses the degree of similarity between sequences. Two sequences are *homologous* if they share a common ancestor, which is not always easy to determine. The degree of similarity obtained by alignment can be useful in determining the possibility of homology between two sequences.

In biology, the sequences to be compared are either nucleotides (DNA, RNA) or amino acids (proteins). In the case of nucleotides, one usually aligns identical nucleotide symbols. When dealing with amino acids the alignment of two amino acids occurs if they are identical or if one can be derived from the other by substitutions that are likely to occur in nature.

An alignment can be either *local* or *global*. In the former, only portions of the sequences are aligned, whereas in the latter one aligns over the entire length of the sequences. Usually, one uses gaps, represented by the symbol "-", to indicate that it is preferable not to align two symbols because in so doing, many other pairs can be aligned. In local alignments there are larger regions of gaps. In global alignments, gaps are scattered throughout the alignment.

A measure of likeness between two sequences is *percent identity*: once an alignment is performed we count the number of columns containing identical symbols. The percent identity is the ratio between that number and the number of symbols in the (longest) sequence. A possible measure or *score* of an alignment is calculated by summing up the matches of identical (or similar) symbols and counting gaps as negative.

With these preliminary definitions in mind, we are ready to describe the algorithms that are often used in sequence comparison.

7.1.1. Pairwise Alignment. Many of the methods of pattern matching used in computer science assume that matches contain no gaps. Thus there is no match for the pattern *bd* in the text *abcd*. In biological sequences, gaps are allowed and an alignment *abcd* with *bd* yields the representation:

$$\begin{array}{cccc} a & b & c & d \\ - & b & - & d. \end{array}$$

Similarly, an alignment of *abcd* with *buc* yields:

$$\begin{array}{ccccc} a & b & - & c & d \\ - & b & u & c & -. \end{array}$$

The above implies that gaps can appear both in the text and in the pattern. Therefore there is no point in distinguishing texts from patterns. Both are called sequences. Notice that, in the above examples, the alignments maximize matches of identical symbols in both sequences. Therefore, sequence alignment is an optimization problem.

A similar problem exists when we attempt to automatically correct typing errors like character replacements, insertions, and deletions. Google and Word, for example, are able to handle some typing errors and display suggestions for possible corrections. That implies searching a dictionary for best matches.

An intuitive way of aligning two sequences is by constructing dot matrices. These are Boolean matrices representing possible alignments that can be detected visually. Assume that the symbols of a first sequence label the rows of the Boolean matrix and those of the second sequence label the columns. The matrix is initially set to *zero* (or *false*). An entry becomes a *one* (or *true*) if the labels of the corresponding row and column are identical.

Consider now two identical sequences. Initially, assume that all the symbols in a sequence are different. The corresponding dot matrix has *ones* in the diagonal indicating a perfect match. If the second sequence is a subsequence of the first, the dot matrix also shows a shorter diagonal line of *ones* indicating where matches occur.

The usage of dot matrices requires a visual inspection to detect large chunks of diagonals indicating potential common regions of identical symbols. Notice, however, that if the two sequences are long and contain symbols of a small vocabulary (like the four nucleotides in DNA) then noise occurs: that means that there will be a substantial number of scattered *ones* throughout the matrix and there may be several possible diagonals that need to be inspected to find the one that maximizes the number of symbol matches. Comparing multiple symbols instead of just two—one in a row the other in a column—may reduce noise.

An interesting case that often occurs in biology is one in which a sequence contains repeated copies of its subsequences. That results in multiple diagonals, and again visual inspection is used to detect the best matches.

The time complexity of constructing dot matrices for two sequences of lengths m and n is m^*n. The space complexity is also m^*n. These values may well be intolerable for very large values of m and n. Notice also that, if we know that the sequences are very similar, we do not need to build the entire matrix. It suffices to construct the elements around the diagonal. Therefore, one can hope to achieve almost linear complexity in those cases. It will be seen later that the most widely used pairwise sequence alignment algorithm (BLAST) can be described in terms of finding diagonals without constructing an entire dot matrix.

The dot matrix approach is useful but does not yield precise measures of similarity among sequences. To do so, we introduce the notion of costs for the gaps, exact matches, and the fact that sometimes an alignment of different symbols is tolerated and considered better than introducing gaps.

We will now open a parenthesis to motivate the reader of the advantages of using dynamic programming in finding optimal paths in graphs. Let us consider a directed acyclic graph (DAG) with possibly multiple entry nodes and a single exit node. Each directed edge is labeled with a number indicating the cost (or weight) of that edge in a path from the entry to the exit node. The goal is to find an optimal (maximal or minimal) path from an entry to the exit node.

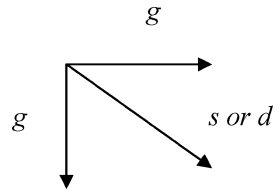
The dynamic programming (DP) approach consists of determining the best path to a given node. The principle is simple: consider all the incoming edges e_i to a node V. Each of these edges is labeled by a value v_i indicating the weight of the edge e_i. Let p_i be the optimal values for the nodes that are the immediate predecessors of V. An optimal path from an entry point to V is the one corresponding to the maximum (or the minimum) of the quantities: $p_1 + v_1, p_2 + v_2, p_3 + v_3, \ldots$ etc.

Starting from the entry nodes one determines the optimal path connecting that node to its successors. Each successor node is then considered and processed as above. The time complexity of the DP algorithm for DAGs is linear with its number of nodes. Notice that the algorithm determines a single value representing the total cost of the optimal path.

If one wished to determine the sequence of nodes in that path, then one would have to perform a second (backward) pass starting from the exit node and retrace one by one the nodes in that optimal path. The complexity of the backward pass is also linear. A word of caution is in order. Let us assume that there are several paths that yield the same total cost. Then the complexity of the backward pass *could be exponential!*

We will now show how to construct a DAG that can be used to determine an optimal pairwise sequence alignment. Let us assume that the cost of introducing a gap is g, the cost of matching two identical symbols is s,and the choice of tolerating the alignment of different symbols is d. In practice when matching nucleotide sequences it is common to use the weights $g = -2, s = 1$, and $d = -1$. That attribution of weights penalizes gaps most, followed by a tolerance of unmatched symbols; identical symbols induce the highest weight. The optimal path being sought is the one with a total maximum cost.

Consider now the following *part* of a DAG expressing the three choices dealing with a pair of symbols being aligned.



The horizontal and vertical arrows state that a gap may be introduced either in the top or in the bottom sequences. The diagonal indicates that the symbols will be

aligned and the cost of that choice is either s (if the symbols match) or d if they do not match.

Consider now a two-dimensional matrix organized as the previously described dot matrix, with its rows and columns being labeled by the elements of the sequences being aligned. The matrix entries are the nodes of a DAG, each node having the three outgoing directed edges as in the above representation. In other words the matrix is tiled with copies of the above subgraph. Notice that the bottommost row (and the rightmost column) consists only of a sequence of directed horizontal (vertical) edges labeled by g.

An optimal alignment amounts to determining the optimal path in the overall DAG. The single entry node is the matrix element with indices [0,0] and the single exit node is the element indexed by $[m, n]$ where m and n are the lengths of the two sequences being aligned.

The DP measure of similarity of the pairwise alignment is a score denoting the sum of weights in the optimal path. With the weights mentioned earlier ($g = -2, s = 1$, and $d = -1$), the best score occurs when two identical sequences of length n are aligned; the resulting score is n, since the cost attributed to a diagonal edge is 1. The worst (unrealistic) case occurs when aligning a sequence with the empty sequence resulting in a score of $-2n$. Another possible outcome is that different symbols become aligned since the resulting path yields better scores than those that introduce gaps. In that case, the score becomes $-n$.

Now a few words about the complexity of the DP approach for alignment. Since there are $m * n$ nodes in the DAG, the time and space complexity is quadratic when the two sequences have similar lengths. As previously pointed out, that complexity can become intolerable (exponential) if there exist multiple optimal solutions and all of them need to be determined. (That occurs in the backward pass.)

The two often-used algorithms for pairwise alignment are those developed by the pairs of co-authors Needleman–Wunsch and Smith–Waterman. They differ on the costs attributed to the outmost horizontal and vertical edges of the DAG. In the Needleman–Wunsch approach, one uses weights for the outmost edges that encourage the best overall (global) alignment. In contrast, the Smith–Waterman approach favors the contiguity of segments being aligned.

Most of the textbooks mentioned in the references (e.g., Setubal and Meidanis [1997], Durbin et al. [1998], and Dwyer [2002]) contain good descriptions of using dynamic programming for performing pairwise alignments.

7.1.2. Aligning Amino Acids Sequences. The DP algorithm is applicable to any sequence provided the weights for comparisons and gaps are properly chosen. When aligning nucleotide sequences the previously mentioned weights yield good results. A more careful assessment of the weights has to be done when aligning sequences of amino acids. This is because the comparison between any two amino acids should take evolution into consideration.

Biologists have developed 20×20 triangular matrices that provide the weights for comparing identical and different amino acids as well as the weight that should be attributed to gaps. The two more frequently used matrices are known as PAM (Percent Accepted Mutation) and BLOSUM (Blocks Substitution Matrix). These matrices reflect the weights obtained by comparing the amino acids substitutions that have occurred through evolution. They are often called substitution matrices.

One usually qualifies those matrices by a number: the higher values of the X in either PAM X or BLOSUM X, indicate more lenience in estimating the difference between two amino acids. An analogy with the previously mentioned weights clarifies what is meant by lenience: a weight of 1 attributed to identical symbols and 0 attributed to different symbols is more lenient than retaining the weight of 1 for symbol identity and utilizing the weight -1 for nonidentity.

Many bioinformatics texts (e.g., Mount [2001] and Pevzner [2000]) provide detailed descriptions on how substitution matrices are computed.

7.1.3. Complexity Considerations and BLAST. The quadratic complexity of the DP-based algorithms renders their usage prohibitive for very large sequences. Recall that the present genomic database contains about 30 billion base pairs (nucleotides) and thousands of users accessing that database simultaneously would like to determine if a sequence being studied and made up of thousands of symbols can be aligned with existing data. That is a formidable problem!

The program called BLAST (Basic Local Alignment Search Tool) developed by the National Center for Biotechnology Information (NCBI) has been designed to meet that challenge. The best way to explain the workings of BLAST is to recall the approach using dot matrices. In BLAST the sequence, whose presence one wishes to investigate in a huge database, is split into smaller subsequences. The presence of those subsequences in the database can be determined efficiently (say by hashing and indexing).

BLAST then attempts to pursue further matching by extending the left and right contexts of the subsequences. The pairings that do not succeed well are abandoned and the best match is chosen as a result of the search. The functioning of BLAST can therefore be described as finding portions of the diagonals in a dot matrix and then attempting to determine the ones that can be extended as much as possible. It appears that such technique yields practically linear complexity. The BLAST site handles about 90,000 searches per day. It success demonstrates that excellent hacking has its place in computer science.

BLAST allows comparisons of either nucleotide or amino acid sequences with those existing in the NCBI database. In the case of amino acids, the user is offered various options for selecting the applicable substitution matrices. Other input parameters are also available.

Among the important information provided by a BLAST search is the p-value associated with each of the sequences that match a user specified sequence. A p-value is a measure of how much evidence we have against the null hypotheses. (The null hypothesis is that observations are purely the result of chance.) A very small p-value (say of the order of 10^{-19}) indicates that it is very unlikely that the sequence provided by the search is totally unrelated to the one provided by the user. The home page of BLAST is an excellent source for a tutorial and a wealth of other information (http://www.ncbi.nlm.nih.gov/BLAST/).

FASTA is another frequently used search program with a strategy similar to that of BLAST. The differences between BLAST and FASTA are discussed in many texts (e.g., Pevsner [2003]).

A topic that has attracted the attention of present researchers is the comparison between two entire genomes. That involves aligning sequences containing billions of nucleotides. Programs have been developed to handle these time consuming tasks. Among these programs is Pipmaker [Schwartz et al. 2000]; a discussion of the specific problems of comparing two genomes is presented in Miller [2001]. An interesting method of entire genome comparison using suffix trees is described in Delcher et al. [1999].

7.1.4. Multiple Alignments. Let us assume that a multiple alignment is performed for a set of sequences. One calls the *consensus* sequence the one obtained by selecting for each column of the alignment the symbol that appears most often in that column.

Multiple alignments are usually performed using sequences of amino acids that are believed to have similar structures. The biological motivation for multiple alignments is to find common patterns that are conserved among all the sequences being considered. Those patterns may elucidate aspects of the structure of a protein being studied.

Trying to extend the dot matrix and DP approaches to the alignment of three

or more sequences is a tricky proposition. One soon gets into difficult time-consuming problems. A three-dimensional dot matrix cannot be easily inspected visually. The DP approach has to consider DAGs whose nodes have seven outgoing edges instead of the three edges needed for pairwise alignment (see, e.g., Dwyer [2002]).

As dimensionality grows so does algorithmic complexity. It has been proved that multiple alignments have exponential complexity with the number of sequences to be aligned. That does not prevent biologists from using approximate methods. These approximate approaches are sometimes unsatisfactory; therefore, multiple alignment remains a worthy topic of research. Among the approximate approaches, we consider two.

The first is to reduce a multiple alignment to a series of pairwise alignments and then combine the results. One can use the DP approach to align all pairs of sequences and display the result in a triangular matrix form such that each entry $[i, j]$ represents the score obtained by aligning sequence i with sequence j.

What follows is more an art than science. One can select a center sequence C as the one that yields a maximum sum of pairwise scores with all others. Other sequences are then aligned with C following the empirical rule: once a gap is introduced it is never removed. As in the case of pairwise alignments, one can obtain global or local alignments. The former attempts to obtain an alignment with maximum score regardless of the positions of the symbols. In contrast, local alignments favor contiguity of matched symbols.

Another approach for performing multiple alignments is using the Hidden Markov Models (HMMs), which are covered in Section 7.3.2.

CLUSTAL and its variants are software packages often used to produce multiple alignments. As in the case of pairwise alignments these packages offer capabilities of utilizing substitution matrices like BLOSUM or PAM. A description of CLUSTAL W appears in Tompson et al. [1994].

7.1.5. Pragmatic Aspects of Alignments. An important aspect of providing results for sequence alignments is their presentation. Visual inspection is crucial in obtaining meaningful interpretations of those results. The more elaborate packages that perform alignments use various colors to indicate regions that are conserved and provide statistical data to assess the confidence level of the results.

Another aspect worth mentioning is the variety of formats that are available for input and display of sequences. Some packages require specific formats and, in many cases, it is necessary to translate from one format to another.

7.2. Phylogenetic Trees

Since evolution plays a key role in biology, it is natural to attempt to depict it using trees. These are referred to as phylogenetic trees: their leaves represent various organisms, species, or genomic sequences; an internal node P stands for an abstract organism (species, sequence) whose existence is presumed and whose evolution led to the organisms whose direct descendants are the branches emanating from P.

A motivation for depicting trees is to express—in graphical form—the outcome of multiple alignments by the relationships that exist between pairs or groups of sequences. These trees may reveal evolutionary inconsistencies that have to be resolved. In that sense the construction of phylogenetic validates or invalidates conjectures made about possible ancestors of a group of organisms.

Consider a multiple alignment: Among its sequences one can select two, whose pairwise score yields the highest value. We then create an abstract node representing the direct ancestor of the two sequences.

A tricky step then is to reconstruct—among several possible sequences—one that best represents its children. This requires both ingenuity and intuition. Once the choice is made, the abstract ancestor sequence replaces its two children and the algorithm continues recursively until a root node is determined. The result is a binary tree whose root represents the

primordial sequence that is assumed to have generated all the others. We will soon revisit this topic.

There are several types of trees used in bioinformatics. Among them, we mention the following:

(1) *Unrooted* trees are those that specify distances (differences) between species. The length of a path between any two leaves represents the accumulated differences.

(2) *Cladograms* are *rooted trees* in which the branches' lengths have no meaning; the initial example in this section is a cladogram.

(3) *Phylograms* are extended cladograms in which the length of a branch quantifies the number of genetic transformations that occurred between a given node and its immediate ancestor.

(4) *Ultrametric* trees are phylograms in which the accumulated distances from the root to each of the leaves is quantified by the same number; ultrametric trees are therefore the ones that provide most information about evolutionary changes. They are also the most difficult to construct.

The above definitions suggest establishing some sort of molecular clock in which mutations occur at some predictable rate and that there exists a linear relationship between time and number of changes. These rates are known to be different for different organisms and even for the various cell components (e.g., DNA and proteins). That shows the magnitude and difficulty of establishing correct phylogenies.

An algorithm frequently used to construct unrooted trees is called UPGMA (for Unweighted Pair Group Method using Arithmetic averages). Let us reconsider the initial example of multiple alignments and assume that one can quantify the distances between any two pairwise alignments (The DP score of those alignments could yield the information about distances: the higher the score, the lower is the distance among the sequences). The various distances can be summarized in triangular matrix form.

The UPGMA algorithm is similar to the one described in the initial example. Consider two elements E_1 and E_2 having the lowest distance among them. They are grouped into a new element (E_1, E_2). An updated matrix is constructed in which the new distances to the grouped elements are the averages of the previous distances to E_1 and E_2. The algorithm continues until all nodes are collapsed into a single node. Note that if the original matrix contains many identical small entries there would be multiple solutions and the results may be meaningless. In bioinformatics—as in any field—one has to exercise care and judgment in interpreting program output.

The notion of *parsimony* is often invoked in constructing phylograms, rooted trees whose branches are labeled by the number of evolutionary steps. Parsimony is based on the hypothesis that mutations occur rarely. Consequently, the overall number of mutations assumed to have occurred throughout evolutionary steps ought to be minimal. If one considers the change of a single nucleotide as a mutation, the problem of constructing trees from sequences becomes hugely combinatorial.

An example illustrates the difficulty. Let us assume an unrooted tree with two types of labels. Sequences label the nodes and numbers label the branches. The numbers specify the mutations (symbol changes within a given position of the sequence) occurring between the sequences labeling adjacent nodes.

The problem of constructing a tree using parsimony is: given a small number of short nucleotide sequences, place them in the nodes and leaves of an unrooted tree so that the overall number of mutations is minimal.

One can imagine a solution in which all possible trees are generated, their nodes labeled and the tree with minimal overall mutations is chosen. This approach is of course forbidding for large sets of long sequences and it is another example of the ubiquity of difficult combinatorial optimization problems in bioinformatics. Mathematicians and theoretical computer scientists have devoted considerable effort

in solving efficiently these types of problems (see, e.g., Gusfield [1997]).

We end this section by presenting an interesting recent development in attempting to determine the evolutionary trees for entire genomes using data compression [Bennett et al. 2003]. Let us assume that there is a set of long genomic sequences that we want to organize as a cladogram. Each sequence often includes a great deal of intergenic (noncoding) DNA material whose function is still not well understood. The initial example in this section is the basis for constructing the cladogram.

Given a text T and its compressed form C, the ratio $r = |C|/|T|$ (where $|\alpha|$ is the length of the sequence α) expresses the degree of compression that has been achieved by the program. The smaller the ratio the more compressed is C. Data compressors usually operate by using dictionaries to replace commonly used words by pointers to words in the dictionary.

If two sequences $S1$ and $S2$ are very similar, it is likely that their respective r ratios are close to each other. Assume now that all the sequences in our set have been compressed and the ratios r are known. It is then easy to construct a triangular matrix whose entries specify the differences in compression ratios between any two sequences.

As in the UPGMA algorithm, we consider the smallest entry and construct the first binary node representing the pair of sequences S_i and S_j that are most similar. We then update the matrix by replacing the rows corresponding to S_i and S_j by a row representing the combination of S_i with S_j. The compression ratio for the combined sequences can be taken to be the average between the compression ratios of S_i and S_j . The algorithm proceeds as before by searching for the smallest entry and so on, until the entire cladogram is constructed.

An immense advantage of the data compression approach is that it will consider as very similar two sequences $\alpha\beta\gamma\delta$ and $\alpha\delta\gamma\beta$, where α, β, γ, and δ are long subsequences that have been swapped around. This is because the two sequences are likely to have comparable compression ratios. Genome rearrangements occur during evolution and could be handled by using the data compression approach.

Finally, we should point out the existence of horizontal transfers in molecular biology. This term implies that the DNA of given organism can be modified by the inclusion of foreign DNA material that cannot be explained by evolutionary arguments. That occurrence may possibly be handled using the notion of similarity derived from data compression ratios.

A valuable reference on phylogenetic trees is the recent text by Felsenstein [2003]. It includes a description of PHYLIP (Phylogenetic Inference Package), a frequently used software package developed by Felsenstein for determining trees expressing the relationships among multiple sequences.

7.3. Finding Patterns in Sequences

It is frequently the case in bioinformatics that one wishes to delimit parts of sequences that have a biological meaning. Typical examples are determining the locations of promoters, exons, and introns in RNA, that is, gene finding, or detecting the boundaries of α-helices, β-sheets, and coils in sequences of amino acids. There are several approaches for performing those tasks. They include neural nets, machine learning, and grammars, especially variants of grammars called probabilistic [Wetherell 1980].

In this subsection, we will deal with two of such approaches. One is using grammars and parsing. The other, called Hidden Markov Models or HMMs, is a probabilistic variant of parsing using finite-state grammars.

It should be remarked that the recent capabilities of aligning entire genomes (see Section 7.1.3) also provides means for gene finding in new genomes: assuming that all the genes of a genome G_1 have been determined, then a comparison with the genome G_2 should reveal likely positions for the genes in G_2.

7.3.1. Grammars and Parsing. Chomsky's language theory is based on grammar

rules used to generate sentences. In that theory, a nonterminal is an identifier naming groups of contiguous words that may have subgroups identified by other nonterminals.

In the Chomsky hierarchy of grammars and languages, the finite-state (FS) model is the lowest. In that case, a nonterminal corresponds to a state in a finite-state automaton. In context-free grammars one can specify a potentially infinite number of states. Context-free grammars (CFG) allow the description of palindromes or matching parentheses, which cannot be described or generated by finite-state models.

Higher than the context-free languages are the so-called context sensitive ones (CSL). Those can specify repetitions of sequence of words like ww, where w is any sequence using a vocabulary. These repetitions cannot be described by CFGs.

Parsing is the technique of retracing the generation of a sentence using the given grammar rules. The complexity of parsing depends on the language or grammar being considered. Deterministic finite-state models can be parsed in linear time. The worst case parsing complexity of CF languages is cubic. Little is known about the complexity of general CS languages but parsing of its strings can be done in finite time.

The parse of sentences in a finite-state language can be represented by the sequence of states taken by the corresponding finite-state automaton when it scans the input string. A tree conveniently represents the parse of a sentence in a context-free language. Finally, one can represent the parse of sentence in a CSL by a graph. Essentially, an edge of the graph denotes the symbols (or nonterminals) that are grouped together.

This prelude is helpful in relating language theory with biological sequences. Palindromes and repetitions of groups of symbols often occur in those sequences and they can be given a semantic meaning. Searls [1992, 2002] has been a pioneer in relating linguistics to biology and his papers are highly recommended.

All the above grammars, including finite-state, can generate ambiguous strings, and ambiguity and nondeterminism are often present when analyzing biological sequences. In ambiguous situations—as in natural language—one is interested in the most-likely parse. And that parse can be determined by using probabilities and contexts. In biology, one can also use energy considerations and dynamic programming to disambiguate multiple parses of sequences.

There is an important difference between linguistics as used in natural language processing and linguistics applied to biology. The sentences or speech utterances in natural language usually amount to a relatively few words. In biology, we have to deal with thousands!

It would be wonderful if we could produce a grammar defining a *gene*, as a nonterminal in a language defining DNA. But that is an extremely difficult task. Similarly, it would be very desirable to have a grammar expressing protein folds. In that case, a nonterminal would correspond to a given structure in 3D space. As in context-sensitive languages, the parse (a graph) would indicate the subcomponents that are close together.

It will be seen later (Section 7.4.1) that CFGs can be conveniently used to map RNA sequences into 2D structures. However, it is doubtful that practical grammars would exist for detecting genes in DNA or determining tertiary structure of proteins.

In what follows, we briefly describe the types of patterns that are necessary to detect genes in DNA. A nonterminal G, defined by the rules below, can roughly describe the syntax of genes:

$$
\begin{aligned}
G &\rightarrow PR \\
P &\rightarrow N \\
R &\rightarrow EIR|E \\
E &\rightarrow N \\
I &\rightarrow gtNag,
\end{aligned}
$$

where N denotes a sequence of nucleotides a , c , g , t ; E is an exon, I an intron, R a

sequence of alternating exons and introns and P is a promoter region, that is, a heading announcing the presence of the gene. In this simplified grammar, the markers gt and ag are delimiters for introns.

Notice that it is possible to transform the above CFG into an equivalent FSG since there is a regular expression that defines the above language. But the important remark is that the grammar is highly ambiguous since the markers gt or ag could appear anywhere within an exon an intron or in a promoter region. Therefore, the grammar is descriptive but not usable in constructing a parser.

One could introduce further constraints about the nature of promoters, require that the lengths of introns and exons should adhere to certain bounds, and that the combined lengths of all the exons should be a multiple of three since a gene is transcribed and spliced to form a sequence of triplets (codons).

Notice that even if one transforms the above rules with constraints into finite-state rules, the ambiguities will remain. The case of alternative splicing bears witness to the presence of ambiguities. The alternation exons-introns can be interpreted in many different ways, thus accounting for the fact that a given gene may generate alternate proteins depending on contexts. Furthermore, in biology there are exceptions applicable to most rules. All this implies that probabilities will have to be introduced. This is a good motivation for the need of probabilistic grammars as shown in the following section.

7.3.2. Hidden Markov Models (HMMs). HMMs are widely used in biological sequence analysis. They originated and still play a significant role in speech recognition Rabiner [1989].

HMMs can be viewed as variants of *probabilistic or stochastic* finite-state transducers (FSTs). In an FST, the automaton changes states according to the input symbols being examined. On a given state, the automaton also outputs a symbol. Therefore, FSTs are defined by sets of states, transitions, and input and output vocabularies. There is as usual an initial state and one or more final states.

In a probabilistic FST, the transitions are specified by probabilities denoting the chance that a given state will be changed to a new one upon examining a symbol in the input string. Obviously, the probabilities of transition emanating from a given state for a given symbol have to add up to 1. The automata that we are dealing with *can be and usually are nondeterministic.* Therefore, upon examining a given input symbol, the transition depends on the specified probabilities.

As in the case of an FST, an output is produced upon reaching a new state. An HMM is a probabilistic FST in which there is also a set of pairs $[p, s]$ associated to each state; p is a probability and s is a symbol of the output vocabulary. The sum of the p's in each set of pairs within a given state also has to equal 1. One can assume that the input vocabulary for an HMM consists of a unique dummy symbol (say, the equivalent of an empty symbol). Actually, in the HMM paradigm, we are solely interested in state transitions and output symbols. As in the case of finite-state automata, there is an initial state and a final state.

Upon reaching a given state, the HMM automaton produces the output symbol s with a probability p. The p's are called emission probabilities. As we described so far, the HMM behaves as a string generator.

The following example inspired from Durbin et al. [1998] is helpful to understand the algorithms involved in HMMs.

Assume we have two coins: one, which is unbiased, the other biased. We will use the letters F (fair) for the former and L (loaded) for the latter. When tossed, the L coin yields *Tails* 75% of the time. The two coins are indistinguishable from each other in appearance.

Now imagine the following experiment: the person tossing the coins uses only one coin at a given time. From time to time, he alternates between the fair and the crooked coin. However, we do not know at a given time which coin is being used (hence, the term *hidden* in HMM). But let us

assume that the transition probabilities of switching coins are known. The transition from F to L has probability u, and the transition from L to F has probability v.

Let F be the state representing the usage of the fair coin and L the state representing the usage of the loaded coin. The emission probabilities for the F state are obviously $1/2$ for *Heads* and $1/2$ for *Tails*. Let us assume that the corresponding probabilities while in L are $3/4$ for *Tails* and $1/4$ for *Heads*.

Let $[r, O]$ denote "emission of the symbol O with probability r" and $\{S, [r, O], w, S'\}$ denote the transition from state S to state S' with probability w. In our particular example, we have:

$$\{F, [1/2, H], u, L\}$$
$$\{F, [1/2, T], u, L\}$$
$$\{F, [1/2, H], 1-u, F\}$$
$$\{F, [1/2, T], 1-u, F\}$$

$$\{L, [3/4, T], v, F\}$$
$$\{L, [1/4, H], v, F\}$$
$$\{L, [3/4, T], 1-v, L\}$$
$$\{L, [1/4, H], 1-v, L\}.$$

Let us assume that both u and v are small, that is, one rarely switches from one coin to another. Then the outcome of a generator simulating the above HMM could produce the string:

$$\ldots HTHTTHTT \ldots\ldots HTTTTTTTHT \ldots$$
$$\ldots FFFFFFFF \ldots\ldots LLLLLLLLLL \ldots.$$

The sequence of states below each emitted symbol indicates the present state F or L of the generator.

The main usage of HMMs is in the reverse problem: recognition or parsing. Given a sequence of H's and T's, attempt to determine *the most likely* corresponding state sequence of F's and L's.

We now pause to mention an often-neglected characteristic of nondeterministic and even ambiguous finite-state-automata (FSA). Given an input string accepted by the automaton, it is possible to generate a directed acyclic graph (DAG) expressing all possible parses (sequence of transition states). The DAG is a beautifully compact form to express all those parses. The complexity of the DAG construction is $O(n^*|S|\ )$ in which n is the size of the input string and $|S|$ is the number of states. If $|S|$ is small, then the DAG construction is linear!

Let us return to the biased-coin example. Given an input sequence of H's and T's produced by an HMM generator and also the numeric values for the transition and emission probabilities, we could generate a labeled DAG expressing all possible parses for the given input string. The label of each edge and node of the graph correspond to the transition and emission probabilities.

To determine the optimal parse (path), we are therefore back to dynamic programming (DP) as presented in the section on alignments (7.1). The DP algorithm that finds the path of maximum likelihood in the DAG is known as the Viterbi algorithm: given an HMM and an input string it accepts, determine the most likely parse, that is, the sequence of states that best represent the parsing of the input string.

A biological application closely related to the coin-tossing example is the determination of GC islands in DNA. Those are regions of DNA in which the nucleotides G and C appear in a higher frequency than the others. The detection of GC islands is relevant since genes usually occur in those regions.

An important consideration not yet discussed is how to determine the HMMs transition and emission probabilities. To answer that question, we have to enter the realm of machine learning.

Let us assume the existence of a learning set in which the sequence of tosses is annotated by very good guesses of when the changes of coins occurred. If that set is available one can compute the transition and emission probabilities simply by ratios of counts. This learning is referred as supervised learning since the user provides the answers (states) to each sequence of tosses.

An even more ambitious question is: can one determine the probabilities without having to annotate the input string? The answer is *yes,* with reservations. First, one has to suspect the existence of different states and the topology of the HMM; furthermore, the generated probabilities may not be of practical use if the data is noisy. This approach is called unsupervised learning and the corresponding algorithm is called Baum—Welch.

The interested reader is highly recommended to consult the book by Durbin et al. [1998] where the algorithms of Viterbi and Baum—Welch (probability generator) are explained in detail. A very readable, paper by Krogh [1998] is also advocated. That paper describes interesting HMMs applications such as multiple alignments and gene finding. Many applications of HMMs in bioinformatics consist of finding subsequences of nucleotides or amino acids that have biological significance. These include determining promoter or regulatory sites, and protein secondary structure.

It should be remarked that there is also a theory for stochastic context-free-grammars; those grammars have been used to determine RNA structure. That topic is discussed in the following section.

7.4. Determining Structure

From the beginning of this article, we reminded the reader of the importance of structure in biology and its relation to function. In this section, we review some of the approaches that have been used to determine 3D structure from linear sequences. A particular case of structure determination is that of RNA, whose structure can be approximated in two dimensions. Nevertheless, it is known that 3D knot-like structures exist in RNA.

This section has two subsections. In the first, we cover some approaches available to infer 2D representations from RNA sequences. In the second, we describe one of the most challenging problems in biology: the determination of the 3D structure of proteins from sequences of amino acids. Both problems deal with minimizing energy functions.

7.4.1. RNA Structure. It is very convenient to describe the RNA structure problem in terms of parsing strings generated by context-free-grammars (CFG). As in the case of finite-state automata used in HMMs we have to deal with highly ambiguous grammars. The generated strings can be parsed in multiple ways and one has to choose an optimal parse based on energy considerations.

RNA structure is determined by the attractions among its nucleotides: A (adenine) attracts U (uracil) and C (cytosine) attracts G (guanine). These nucleotides will be represented using small case letters. The CFG rules:

$$S \rightarrow aSu/uSa/\varepsilon$$

generate palindrome-like sequences of u's and a's of even length. One could map this palindrome to a 2D representation in which each a in the left of the generated string matches the corresponding u in the right part of the string and vice-versa. In this particular case, the number of matches is maximal.

This grammar is nondeterministic since a parser would not normally know where lies the middle of the string to be parsed. The grammar becomes highly ambiguous if we introduce a new nonterminal N generating any sequence of a's and u's.

$$S \rightarrow aSu/uSa/N \qquad N \rightarrow aN/uN/\varepsilon.$$

Now the problem becomes much harder since any string admits a very a large number of parses and we have to chose among all those parses the one that matches most a's with u's and vice versa. The corresponding 2D representation of that parse is what is called a *hairpin loop.*

The parsing becomes even more complex if we introduce the additional rule:

$$S \rightarrow SS.$$

That rule is used to depict bifurcations of RNA material. For example, two hairpin

structures may be formed, one corresponding to the first S, the second to the second S. The above rule increases exponentially the number of parses.

An actual grammar describing RNA should also include the rules specifying the attractions among c's and g's:

$$S \rightarrow cSg/gSc/.$$

And N would be further revised to allow for left and right bulges in the 2D representation. These will correspond to left and right recursions for the new rules defining N:

$$N \rightarrow aN/uN/cN/gN/Na/Nu/Nc/Ng/\varepsilon.$$

The question remains that, from all parses, we have to select the one yielding the maximal number of complementary pairs. And there could be several enjoying that property.

Zuker has developed a clever algorithm, using DP, that is able to find the best parse in n^3 time where n is the length of the sequence (see Zuker and Stiegler [1981]). That is quite an accomplishment since just the parsing of strings generated by general (ambiguous) CFG is also cubic.

Recent work by Rivas and Eddy [2000] shows that one can use variants of context sensitive grammars to map RNA sequences onto structures containing knots, that is, overlaps that actually make the structure three-dimensional. That results in higher than cubic complexity.

We should point out that the cubic complexity is acceptable in natural language processing or in speech recognition where the sentences involved are relatively short. Determining the structure of RNA strings involving thousands of nucleotides would imply in unbearable computation times.

One should also keep in mind that multiple solutions in the vicinity of a theoretical optimum may well exist; some of those may be of interest to biologists and portray better what happens in nature. Ideally, one would want to introduce constraints and ask questions like: Given an RNA string and a 2D pattern constrained to satisfy a given geometrical criteria, is there an RNA configuration that exhibits that 2D pattern and is close to the optimum?

We end this subsection by pointing out a worthy extension of CFGs called stochastic or probabilistic CFG's. Recall from Section 7.3.2 that HMMs could be viewed as probabilistic finite-state transducers. Stochastic CFGs have been proposed and utilized in biology (see Durbin et al. [1998]). Ideally, one would like to develop the counterparts of the Viterbi and Baum–Welch algorithms applicable to stochastic CFGs and that topic is being investigated. This implies that the probabilities associated to a given CFG could be determined by a learning set, in a manner similar to that used to determine probabilities for HMMs.

7.4.2. Protein Structure. We have already mentioned the importance of 3D structures in biology and the difficulty in obtaining the actual 3D structures for proteins described by a sequence of amino acids. The largest repository of 3D protein structures is the PDB (Protein Data Base): it records the actual x, y, z coordinates of each atom making up each of its proteins. That information has been gathered mostly by X-ray crystallography and NMR techniques.

There are very valuable graphical packages (e.g., Rasmol) that can present the dense information in the PDB in a visually attractive and useful form allowing the user to observe a protein by rotating it to inspect its details viewed from different angles.

The outer surface of a protein consists of the amino acids that are *hydrophilic* (tolerate well the water media that surrounds the protein). In contrast, the *hydrophobic* amino acids usually occupy the protein's core. The configuration taken by the protein is one that minimizes the energy of the various attractions and repulsions among the constituent atoms.

Within the 3D representation of a protein, one can distinguish the following

components. A *domain* is a portion of the protein that has its own function. Domains are capable of independently folding into a stable structure. The combination of domains determines the protein's function.

A *motif* is a generalization of a short pattern (also called *signature* or *fingerprint*) in a sequence of amino acids, representing a feature that is important for a given function. A motif can be defined by regular expressions involving the concatenation, union, and repetition of amino acid symbols. Since function is the ultimate objective in the study of proteins, both domains and motifs are used to characterize function.

In what follows, we will present the bioinformatics approaches that are being used to describe and determine 3D protein structure. We mentioned in Section 7.3 that there exist several approaches that attempt to determine secondary structure of proteins by detecting 3D patterns—α-helices, β-sheets, and coils—in a given sequence of amino acids. That detection does not give any information as to how close those substructures are from each other in three-dimensional space.

A team at the EBI (European Bioinformatics Institute) has suggested the use of what is called *cartoons* [Gilbert et al. 1999]. These are two-dimensional representations that express the proximity among components (α-helices and β-sheets).

The cartoon uses graphical conventions—sheets represented by triangles, helices by circles—and lines joining components indicate their 3D closeness. This can be viewed as an extension of the secondary structure notation in which pointers are used to indicate spatial proximity. In essence, cartoons depict the topology of a protein. The EBI group has developed a database with information about cartoons for each protein in the PDB. The objective of the notation is to allow biologists to find groups of combined helices and sheets (domains) that have certain characteristics and function.

Protein folding, the determination of protein structure from a given sequence of amino acids, is one of the most difficult problems in present-day science. The approaches that have been used to solve it can only handle short sequences and require the capabilities of the fastest parallel computers available. (Incidentally, the IBM team that originated the chess-winning program is now developing programs to attempt to solve this major problem.)

Essentially, protein folding can be viewed as an n-body problem as studied by physicists. Assuming that one knows the various attracting and repelling forces among atoms the problem is to find the configuration that minimizes the total energy of the system.

A related approach utilizes lattice models: these assume that the backbone of the protein can be represented by a sequence of edges in mini-cubes packed on a larger cubic volume. In theory, one would have to determine all valid paths within the large cube. This determination requires huge computational resources (see, e.g., Li et al. [1996]). Random walks are often used to generate a valid path and an optimizer computes the corresponding energy; the path is then modified slightly in the search of minimal energy configurations. As in many problems of this kind, optimizers try to avoid local minima.

The above brute-force approaches may be considered as long-term efforts requiring significant investment in computer equipment. The more manageable present formulations often use what is called the *inverse* protein-folding problem: given a known 3D structure S of a protein corresponding to a sequence, attempt to find all other sequences that will fold in a manner similar to S. As mentioned earlier (Section 2) structure similarity does not imply sequence similarity.

An interesting approach called *threading* is based on the inverse protein paradigm. Given a sequence of amino acids, a threading program compares it with all the existing proteins in the PDB and determines a possible variant of the PDB protein that best matches the one being considered.

More details about threading are as follows: Given a sequence s, one initially determines a variant of its secondary structure T defined by intervals within s where each possible helix or sheet may occur; let us refer to helices and sheets simply as components. The threading program uses those intervals and an energy function E that takes into account the proximity of any pair of components. It then uses branch-and-bound algorithms to minimize E and determine the most likely boundaries between the components [Lathrop and Smith 1996]. A disadvantage of the threading approach is that it cannot discover new folds (structures).

There are several threading programs available in the Web (*Threader* being one of them). These programs are given an input sequence s and provide a list of all the structures S in the PDB that are good "matches" for s.

There are also programs that match 3D structures. For example, it is often desirable to know if a domain of a protein appears in other proteins.

Protein structure specialists have an annual competition (called CASP for *Critical Assessment of Techniques for Protein Structure Prediction*) in which the participant teams are challenged to predict the structure of a protein given by its amino acid sequence. That protein is one whose structure has been recently determined exactly by experiments but is not yet available at large. The teams can use any of the available approaches.

In recent years, there has been some success with the so-called ab initio techniques. They consist of initially predicting secondary structure and then attempting to position the helices and sheets in 3D so as to minimize the total energy. This differs from threading in the sense that all possible combinations of proximity of helices and sheets are considered in the energy calculations. (Recall that in threading intervals are provided to define the boundaries of helices and sheets.) One can think of ab initio methods as those that place the linkages among the components of the above mentioned *cartoons*.

7.5. Cell Regulation

In this section, we present two among the existing approaches to simulate and model gene interaction. The terms *simulation* and *modeling* are usually given different meanings. A simulation mimics genes' interactions and produces results that can be compared with actual experimental data to check if the model used in the simulation is realistic. In modeling the experimental data is provided and one is asked to provide the model. Modeling is a reverse engineering problem that is much harder than simulation. Modeling is akin to program synthesis from data.

Although, in this section, we only deal with gene interactions, the desirable outcome of regulation research is to produce micro-level flowcharts representing metabolic and signaling pathways (Section 7.6).

It is important to remark the significance of intergenic DNA material in cell regulation. These regions of noncoding DNA play a key role in allowing RNA-polymerase to start gene transcription. This is because there has to be a suitable docking between the 3-D configurations of the DNA strand and those of the constituents of RNA-polymerase.

7.5.1. Discrete Model. We start by showing how one can easily simulate the interaction of two or more genes by a program involving threads. Consider the following scenario:

Gene *G1* produces protein *P1* in *T1* units of time; *P1* dissipates in time *U1* and triggers condition *C1*. Similarly:

Gene *G2* produces *P2* in *T2* units of time; *P2* dissipates in time *U2* and triggers condition *C2*.

Once produced, *P2* positions itself in *G1* for *U2* units of time preventing *P1* from being produced. We further assume that the production of a protein can only take place if a precondition is satisfied. That precondition is a function of the various post conditions $C_i's$.

The above statements can be presented in program form by assuming the

existence of a procedure *process* involving five parameters:

—The gene identification G (possibly a string)

—A pre-condition C allowing a protein P to be processed (a constraint)

—The units of time T needed to produce protein P

—The time U for protein P to completely dissipate

—A post-condition C' to be performed after the protein is produced (a constraint).

Notice that the precondition C can be a general Boolean function and the post-condition C' can trigger changes in the parameters of any C. A rough description of *process* is:

```
process (Gene, Pre-Condition, Process-Time,
        Decay-Time, Post-Condition)
  if Gene is not available (Pre-condition)
    then wait until it becomes available
    else {produce protein in Process-Time,
          trigger Post-condition,
          wait for the given Decay-Time}
```

The order in which the constituents of the **else** part of the **if**-statement are executed is subject to different interpretations and it is left unspecified.

Now let us imagine that *process* acts like a thread that can be executed in parallel with other threads. We also make the simplifying assumption that the *process* of a given gene G cannot be invoked until the previous incarnation of that process has terminated. Consider the program segment:

```
forever do
  process ("G1", P2 is not on, 50, 20, none)
  process ("G2", none, 200, 50, P2 is on "G1")
```

Let t denote a current time in the execution of the above program. It should be clear to the reader that the behavior of the program can be displayed by successive Boolean vectors $V(t)$ denoting the state *on* or *off* of each of the genes at time t.

The above program is a minuscule example of the type of concurrent processes that take place within the cell. The *processes* can be likened to RNA-polymerases, spliceosomes and ribosomes.

Notice that in the case of eukaryotic cells there would be three levels of cascading processes since different conditions would be applicable to simulate the generation of a given protein. This is because there could be interruptions not only in RNA production, but also in the splicing, and generation of proteins.

Interesting organisms will have thousands of genes and many of them will interact with others in a complex manner that we do not yet know. As mentioned, the state of the program at time t is describable by the vectors $V(t)$. Actually these vectors correspond to information that can be gathered by microarray experiments. Usually microarrays detect not step functions but continuous ones expressing the amount of RNA produced by a cell at a given time under certain conditions.

Now we can state a major and enormously difficult problem in biology: *given the vectors V(t), deduce the pre and post-conditions for a program simulating gene interactions.* This is a reverse engineering problem that is probably undecidable. Nevertheless, we can attempt to solve more manageable problems of the sort: given the results of microarray experiments, is a given conjecture for the *pre* or *post-* conditions possible?

There are groups of computer scientists and biologists working in such problems. One of these groups led by Regev and Shapiro [2002] uses Milner's Pi-calculus to attempt to answer logical questions about conjectures made by biologists. (The Pi-calculus is a formal language for concurrent computational processes, like those used in mobile telephone systems.) Since the results of microarrays are often noisy and uncertain one has to resort to a probabilistic (or stochastic) variant of the Pi-calculus.

Statistical methods like Bayesian networks and support vector machines have also been used in inferring gene behavior from microarray data [Friedman et al. 2000; Brown et al. 2000; Bar-Joseph et al. 2002]. Clustering algorithms (see,

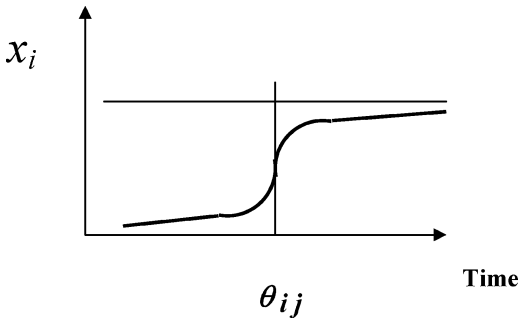


Fig. 1.

e.g., Jain et al. [1999]) are often used to group the genes exhibiting similar behavior, therefore reducing the problem's dimensionality.

7.5.2. Continuous Models. As in the case of the discrete case, we will consider the different aspects: simulation and modeling. The continuous simulation approach is based on the theory of dynamic systems. It is assumed that the expression level of each gene is describable by a differential equation. If there are n genes that interact with each other then the continuous simulation consists of a system of n nonlinear differential equations.

Let x_i denote the expression level of the ith gene. Then the resulting system of differential equations becomes:

$$dx_i/dt = f_i(\boldsymbol{x}) - \gamma_i x_i, x_i \geq 0,$$

where $\boldsymbol{x}$ is the vector $(x_1, x_2, \ldots . x_n)$

The term $-\gamma_i x_i$ states that the concentration of the ith product decreases through spontaneous processes like degradation, diffusion, etc. f_{ij} is the function specifying a combination of sigmoids (highly nonlinear) which describes the interaction between genes i and j; m is a parameter specifying the steepness of the function around θ_{ij} (see Figure 1).

$$f_{ij} = x_j^m / \left(x_j^m + \theta_{ij}^m\right)$$

The above specifies that gene expression increases (or decreases) sharply when a gene interacts with another. It is possible to generate the system of differential equations from a graph whose nodes represent the genes and the branches their interactions. Additionally, the branches can be labeled with +'s or −'s indicating the fact that a gene activates or represses another gene.

Once the graph and the above parameters are known, the system of equations can be generated, solved numerically and yield curves that describe gene expression as a function of time.

These results are the continuous counterparts of those displayed for the discrete simulation described in the previous section. In the discrete case the gene expression was either *on* or *off* whereas in the continuous case the gene expression curves vary smoothly.

The fact remains that it would extremely difficult to do the reverse engineering task of modeling, that is, generating from existing data the system of equations and their parameters. The clustering algorithms mentioned in Section 9 have become indispensable to reduce the complexity of gene regulation analysis from microarray data.

Somogyi and his co-workers [Liang et al. 1998] have proposed an interesting approach for both simulation and modeling of gene interaction. The simulation uses a Boolean approach and the modeling amounts to generating circuits (or equivalent Boolean formulas) from data.

deJong [2002] has recently published an extensive survey about work done in cell regulation both in simulation and in modeling. One interesting way of solving the above differential equations is by qualitative reasoning, a subject developed in artificial intelligence by Kuipers [1994] to deal with discrete versions of differential equations. Cohen [2001] proposes the use of constraints to describe various cell regulation methods.

E-CELL is an ambitious Japanese project that aims at simulating cells by using stochastic systems of nonlinear differential equations [Tomita et al. 1999]. It has been used to simulate the behavior of various cells including that of the human heart. Versions of the E-CELL simulator are available for various platforms.

7.6. Determining Function and Metabolic Pathways

In the previous section, we mentioned that protein domains and motifs where important in determining protein function. Function is a subjective topic that may mean different things for different people. The protein database (PDB) contains annotations—in natural language—that explain the role of the protein in the larger context of cell behavior. Incidentally, great care has to be taken to interpret annotations since different researchers use different terms that are supposed to be equivalent. In a simplistic manner, the function of a protein is its PDB annotation complemented by related observations.

A typical example of an annotations for gene function is as follows: "The gene, known as 5-HTT, has been a focus of depression studies because it contains the code to produce a protein that escorts the chemical messenger serotonin across the spaces between brain cells, or synapses, and then clears away the leftover serotonin" (*New York Times*, July 18, 2003).

The ultimate way to express protein function is by finding its role in metabolic, regulation, and signaling pathways. (These have been briefly defined in the previous sections.) Karp [2001] has studied this topic extensively. He has implemented some of those pathways for *E. coli* and other organisms in the form of databases.

Karp rightfully points out that it is impossible to develop a theory about a complex system without the aid of a properly designed database of facts and interactions among facts. Such database is essentially the representation of large labeled graphs. Each node of the graph represents a chemical reaction, the proteins involved, and the enzymes catalyzing that reaction.

Graphical interfaces are mandatory to display the results of queries about metabolic pathways. For example, one should be able to have graphical responses to questions of the type: (i) determine all the reactions, in which a given enzyme acts as a catalyzer, (ii) find the different enzymes catalyzing similar reactions, (iii) specify all paths going trough a pair of reactions, and so forth. In the *Ecocyc* system developed by Karp, the results of such queries are graph representations with highlighted nodes or paths.

Karp's research is an ambitious one. Ultimately one wants to attempt to generate metabolic pathways from genomic data of similar organisms. The system *Metacyc* is a meta-system developed for that purpose. This type of research should eventually merge with that proposed by Shapiro and briefly described in Section 7.5.1.

The Japanese have also developed a widely used metabolic pathway database called KEGG (Kyoto Encyclopedia of Genes and Genomes) (http://www.genome.ad.jp/kegg/pathway.html).

7.7. Assembling DNA Fragments

The problem of DNA assembly became very important for sequencing very large genomes such as the human genome. Companies like Celera use the so-called *whole genome shotgun* method that consists of sequencing relatively small fragments of DNA and then relying on computer programs to assemble those fragments. Eugene Myers [1999] formerly from Celera, now at Berkeley, has been a pioneer in this effort.

Fragments are of the order of 500 base pairs (bp). The target sequence—the one to be reconstructed—is of the order of 50k to 100k bp, and there are about 1,000 fragments to be assembled.

The problem of assembly becomes complex because of several factors that include *orientation*, *repeats*, and sequencing errors. Fragments can originate from each of the two DNA strands, and orientation means that either a given sequence or its reverse complement is a valid candidate for being assembled into the target sequence. Repeated subsequences in the target sequence make the assembly more difficult because one does know to which copy a given fragment belongs.

A fragment F_i *overlaps* with a fragment F_j if the left (right) end of F_i shares a

common subsequence with the right (left) end of F_j. (If one fragment is a subsequence of another, then the smaller one can be discarded.) A region of contiguously overlapping fragments is called a *contig*.

The assembly problem can be stated as finding the shortest superstring S such that each fragment is a subsequence of S. Remark that this problem bears some similarity with finding multiple alignments. (As we have seen in Section 7.1.4, the latter is known to be a computationally difficult problem.)

It is easily shown that we can construct a labeled directed graph G representing the overlaps of each pair of fragments. Let us assume that F_i overlaps q symbols with F_j and that the length of F_i is greater than the length of F_i. Then the graph G contains a directed edge labeled by q joining the node F_i to the node F_i. Notice that pairwise alignments can be used to determine the edges of the graph and their labels.

It is not difficult to see that a path in G that contains no cycle represents a contig. Therefore the shortest superstring problem amounts to finding the shortest Hamiltonian path in G. That is computationally difficult and one has to resort to approximations. A greedy algorithm is often used to determine that path. The problems of orientation and repeats will also have to be surmounted. A helpful hint is that one knows the approximate size of the target sequence. A recent article by Myers and his colleagues reflects some of the latest work done in DNA assembly [Huson et al. 2002].

A problem related to assembly is that of physical mapping of DNA. The fragments for a given target sequence are obtained from parts of chromosomes containing several hundred thousand base pairs. These very large fragments have markers that enable the reconstruction of the original chromosomal DNA. As in the case of DNA assembly, the reconstruction is based on graphs and again the computational complexity is very high. The reader is directed to the text of Setubal and Meidanis [1997] where that topic is presented pedagogically.

7.8. Using Script Languages

Consider the following typical problem in bioinformatics. Given a sequence of amino acids representing a protein P, we want to use BLAST to determine the proteins in GenBank database that are homologous to P but have a given degree a similarity specified by a p-value threshold. Following that search we may also want to perform a multiple alignment with those homologous proteins (using CLUSTAL) and possibly utilize a package like PHYLIP to determine the phylogenetic tree corresponding to the multiple alignments. Finally, we would like to check if any of the proteins in the multiple alignments has a 3D structure in the PDB.

The cascading use of the above packages would require a researcher to take active part in requesting the URL of a package, performing formats changes if needed, inspecting and rejecting some data, and so forth. Script languages allow their users to write programs that automatically perform these tasks.

Perl and Python are probably the most often-used languages in bioinformatics. Perl is older and has many ready-made packages available for searching websites and downloading results. Python, the more recent language, is gaining momentum in bioinformatics applications.

One of the frequent tasks done using script languages is finding certain patterns in files containing information in various formats (e.g., html). Regular expressions (RE) are often used to specify those set of patterns. A more specific example of RE usage is as follows. Assume that we want to test if a pattern of nucleotides defines the boundaries between exons and introns (these are called *splice-sites*). Also assume that one knows the splice-sites for many genes of a given organism O and can express them by a RE that takes into account the left and right contexts of the splice-sites.

Suppose now that we want to determine the splice-sites for another organism O' that is possibly related to the first. A search using the RE applicable to O may reveal interesting putative splice-sites in

O'. If that is the case, one may wish to revise the RE for O to handle new organisms. These situations occur very often in bioinformatics.

8. A CHIMERICAL PROGRAM

In the previous sections, we reviewed the approaches that are currently being used to solve typical problems in bioinformatics. In this section, we will try to take a glimpse into the future. What follows is the author's extrapolation from current work being done in bioinformatics, and it is admittedly speculative.

The hypothetical program below describes the essence of functional genomics. Given the genome of an organism, it seeks to generate a program that simulates the cell behavior for that organism. At the top level, the program finds all the genes of the genome, then determines the function of each gene, and finally combines the results into a simulator. Paraphrasing in program form one has:

Generation of a cell simulator for a new organism

```
Find-Genes (DNA, Genome)
for each Gene in Genome
    Process (Gene, Function)
    Combine (Function, Cell-Behavior)
```

The first parameter in each of the procedures being called represents either given data or results obtained from a preceding procedure call. *Find-Gene* is probably a version of some of the programs mentioned in Section 7. *Process* embodies the Central Dogma, and *Combine* is admittedly a "pie in the sky" that will have to be worked out in the future. Keep in mind, however, that this is the goal of Karp's project, briefly described in Section 7.6.

In a first phase, *Combine* should generate a program similar to the discrete model example in Section 7.5.1, but involving all genes of the genome. Eventually, one would like to obtain a program that not only mimics gene interactions but also depicts in detail the workings of metabolic and signaling pathways.

An elaborated version of *Process* is given below. It introduces the types of the parameters whenever possible. Even though all the components in the cell are 3D structures, the abstraction of DNA sequences into the type *"string"* is likely to remain applicable. Nevertheless, it is known that the transcription by RNA-polymerase depends on the (elongated) shape of the helix segment that contains the gene.

```
Process (Gene: string,
        Function: program-fragments)
    The central dogma
    RNA-Polymerase (Gene, Pre-
    RNA)
    Note the possibility of
    alternate
    splicing (multiple RNAs)
    Spliceosome (Pre-RNA, RNA)
    Ribosome (RNA, Aminoacid-
    Sequence)
    Fold (Aminoacid-Sequence,
    Structure)
    Determine-Function (Structure,
    Function)
```

The above hypothetical procedure is not unlike the thread *process* described in Section 7.5.1 and leaves open how function can be determined from structure and other data. It is clear that results obtained through microarray experiments, protein interactions, and known metabolic and signaling pathways will have to be taken into consideration.

9. TOPICS THAT PLAY AN IMPORTANT ROLE IN BIOINFORMATICS

A perusal of the material in the previous sections provides insights on CS topics that are likely to influence bioinformatics. A recurring theme in the currently used algorithms is optimization. Alignments, parsimony in phylogeny, determining RNA structure, and protein threading can all be viewed as optimization problems.

The interest in dynamic programming (DP) is that it enables an efficient (polynomial) solution of certain optimization problems. This occurs when a problem can be transformed into determining the maximal (or minimal) path in a DAG. It was seen in the case of pairwise alignments

that one could formulate the problem using a DAG with n^2 nodes, where n is the length of the sequences being aligned. However, the use of DP becomes prohibitive in the case of multiple alignments.

Inevitably, in the case of algorithms with higher complexity one has to resort to heuristics. Typically, heuristic strategies are used in the case of NP problems or polynomial problems involving large volumes of data.

For example, the DNA assembly problem requires suitable heuristics for greedy algorithms to determine possible Hamiltonian paths in a graph. Genetic algorithms have been used for that purpose [Parsons et al. 1995], BLAST illustrates the case in which even a quadratic space and time complexity makes the DP algorithm unusable for practical problems involving huge sequences.

Machine learning, data mining, neural networks, and genetic algorithms occupy a prominent position among the CS approaches used in bioinformatics (see, e.g., Mitchell [1997] and Hand et al. [2000]). This is because there is an enormous amount of data available and, fortunately, biologists have annotated some of this data. Typical examples include gene finding and secondary structure determination (Section 7.3).

There are thousands of genes whose locations in various genomes have been determined using laboratory experiments. This information is recorded in a vast repository of sequences, with markings specifying the locations of promoters, exons, and introns. These annotations enable the determination of the most likely contexts for desired boundaries. The problem becomes: given this learning set, infer the corresponding boundaries for new sequences not in the learning set (supervised learning).

A similar situation occurs when attempting to determine the secondary structure of proteins. An annotated learning set can be obtained from the Protein Data Base (PDB), where thousands of proteins have been studied in detail and for which boundaries of helices and sheets have been accurately determined.

The above are typical problems that can be solved by machine-learning and neural network techniques. Many gene-finders and secondary structure estimators utilize these approaches.

Classification and data clustering (see, e.g., Jain et al. [1999]) are cognate to supervised machine learning. Assume that we are given a large set of lists each containing the values of n parameters and their known classification (say, an identifier). One then groups the lists into clusters that have the same classification.

Given a new list of parameters, we wish to determine the most likely cluster it should belong to. In two-dimensional cases, the answer can be obtained by the evaluation of a simple equation representing the straight line that separates the two semispaces representing the clusters. The n-dimensional case is considered in the relatively new area of support vector machines (SVM). The SVM approach divides the n-dimensional space into areas delimited by semiplanes. These techniques have acquired great significance in reducing the complexity of the task of inferring gene regulation from microarray data.

It is undeniable that probability and statistics play an influential role in bioinformatics. This is not surprising since the data available is huge, varied, and noisy. Recent articles on interpreting microarray experiments utilize statistical approaches such as SVMs and Bayesian networks [Friedman et al. 2000; Brown et al. 2000; Bar-Joseph et al. 2002].

Hidden Markov Models are also machine-learning techniques. In this approach, one starts by specifying a topology of finite-states representing the structure one believes is applicable. Based on the learning set, the probabilities are computed. Given a new sequence, we can then use DP (the Viterbi algorithm) to determine the most likely succession of states corresponding to the given sequence.

All these methods amount to the generation of probabilistic grammars from a learning set. The topology of states in HMMs is generalized to correspond to the presumed grammar rules whose frequency one wishes to estimate. Therefore,

methods using probabilistic grammars are expected to have a salient place in bioinformatics.

Data mining is akin to machine learning. In data mining one hopes to detect certain patterns in huge amounts of data (unsupervised learning). Data mining has been used in forecasting protein interactions [Thierry-Mieg 2000].

It should be apparent that database design and development are integral part of bioinformatics. The best single place to look for information on biological databases is the annual database issue of *Nucleic Acids Research:* http://nar.oupjournals.org/content/vol32/suppl_1/index.shtml. A recommended précis of the problems in the design of genomic and genetic databases and their integration is given in [Ashburner and Goodman 1997].

Computational geometry should also play a key role in analyzing 3D structures. An example is 3D pattern matching in proteins: in this case, the "pattern" is a portion of a protein's backbone, and the "text" corresponds to all the proteins in the PDB. One would want to determine the set of proteins that exhibit that pattern. As in the case of alignments, we would like to tolerate small discrepancies between the pattern and elements in the text.

The example of phylogenetic tree construction using data compression (Section 7.2) illustrates the importance of information theory in analyzing massively long sequences of symbols.

An interesting CS application in bioinformatics is that of natural language processing (NLP). For example, biotech companies hire teams of biologists to examine the large scientific literature available to detect descriptions of possible gene or protein interactions. It would be desirable to automate that process. Another possible NLP application is to attempt to make sense of annotations made by biologists to explain gene function. Questions of the type: *Are two annotations comparable?* are difficult inquiries that one would want to be able to answer.

Graphics and graphical interfaces are of course a necessity for displaying biological data. As in the other CS applications, knowledge of biology and the capacity to interact with biologists are vital to successful software development in bioinformatics.

10. LEARNING MORE ABOUT BIOINFORMATICS

The material covered in this article is but an introduction to the field. The interested reader will have to expand his or her knowledge significantly to become proficient in bioinformatics. A few hints as how to proceed are discussed below.

We dealt with 3D structures in an abstract manner and showed their importance in the molecular interactions that are crucial to cell life. To understand molecular structure and interactions in detail, one has to plunge into biochemistry. Therefore, an introductory course in biochemistry is a prerequisite for doing work in bioinformatics. That and a course in molecular biology are long-term undertakings.

This author favors a continual updating of knowledge by reading the tutorial material available on the Web, and most of all, by interacting with biologists. As mentioned earlier, this is not always an easy task since we have been educated to reason in different modes. Nevertheless, such interactions are necessary in order to infer which tools are best suited to help biologists tackle unsolved problems. And that effort can lead to the development of novel algorithms and approaches.

A recommended introductory bioinformatics text, by Krane and Raymer [2003], has recently been published. It provides an easy to read introduction to the field. A good companion for that book is the *Cartoon Guide to Genetics* by Gonick and Wheelis [1991].

Computer scientists interested in computational biology are referred to the several textbooks currently available that are listed as references. We should distinguish two types of texts: those that emphasize the discrete and combinatorial aspects of the field (e.g., Setubal and Meidanis [1997]), and those that favor a

probabilistic and statistical approach (e.g., Durbin et al. [1998]).

For the reader interested in the research aspects of bioinformatics, the compendium edited by Salzberg et al. [1998] is recommended. The encyclopedic book by Mount [2001], is an excellent reference text. An interesting article by Luscombe et al. [2001] defining the goals of bioinformatics is certainly worth reading. An aperçu of recent advances in bioinformatics appears in Goodman [2002].

Several texts in bioinformatics have been published recently. Among them we note: Dwyer's book stressing programming in bioinformatics using Perl [Dwyer 2002]; a compendium of recent topics in bioinformatics [Orengo et al. 2003]; a practical approach to the field [Claverie and Notredame 2003]; and a treatise on bioinformatics and functional genomics by Pevsner [2003]. Suggestions for implementing bioinformatics undergraduate level courses have appeared in Cohen [2003]. A recent undergraduate text by Jones and Pevzner [2004] is highly recommended.

11. FINAL REMARKS

Searls [1998] rightly pointed out that many current problems, such as those briefly described in Section 7, remain challenging tasks. His list includes: protein structure prediction, homology search, multiple alignment and phylogeny construction, genomic sequence analysis, and gene finding. The most recent developments in biology point in the direction of functional genomics research. That topic not only encompasses Searls' list of challenges but also includes cell simulation and modeling, as well as metabolic pathways.

Nearly all the contents of the present article have been devoted to explaining single cell behavior. The generic-type cell—also called a *stem cell*—can be transformed into any other type of cell that specializes in performing specific functions in a multicellular organism. Blocking the production of certain proteins and encouraging the expression of others achieve this specialization. This process is not yet well understood. Nevertheless, the geographic position of the cell and its neighbors is known to have significant roles as to which genes are turned on and which are switched off.

An interesting article written by computer scientists at MIT deals with the simulation of multiple cells and proposes the paradigm of amorphous computing [Abelson et al. 1995]. It has been inspired by biology, and it develops a massively parallel model that accounts for changes in the shapes of a network of distributed asynchronous computers. This is an example on how biology can be inspirational to computer science. Another prime example is DNA computing, that is, using DNA strands to solve computationally difficult problems.

As to the future of the relationship between computer science and biology, it is worth mentioning an interview given by Knuth [1993]. He argues that major discoveries in computer science are unlikely to occur as frequently as they did in the past few decades. On the other hand, he states that "*Biology easily has 500 years of exciting problems to work on...*".

The accomplishments made in molecular biology in the past half century have been remarkable. Nevertheless, they pale in comparison to the wondrous tasks that lie ahead. Consider, for example, attempting to answer questions like:

—*How do brain cells establish linkages among themselves while an embryo is being formed?*

—*Is it possible to understand better the origins of language and the nature-nurture paradigm?*

—*How does Darwinian evolutionary theory operate at the molecular level?*

These questions pose enormous challenges and Knuth's forecast may even turn out to be conservative.

With the increasing relevance of biology (and bioinformatics) also comes responsibility. In a recent article in the *New York Times*, Kelly [2003], the president of the Federation of American Scientists, points out that a graduate student in biology

using the wet lab (and available bioinformatics tools) could concoct viruses with great potential for harm.

Since the Manhattan Project physicists have been in a similar predicament. It is now the turn of the biologists (and bioinformaticians) to make sure that developments will be used for lofty purposes. For example, understanding the mechanisms of cell differentiation and inferring the gene interactions that produce cancerous cells will no doubt revolutionize medicine.

APPENDIX. A SUMMARY OF CONCEPTS IN MOLECULAR CELL BIOLOGY

The material in this appendix is designed as a concise refresher for the background in molecular cell biology needed to read the main article. Even though we have avoided the description of chemical structures, they are essential to understanding molecular interactions at the atomic level.

There are several detailed texts in cell and molecular biology available. Two often-used ones are those by Lodish et al. [2003] and Alberts et al. [2004]. The reader is also referred to the numerous glossaries and tutorials that exist on the Web. It often suffices to use Google with the desired keywords, followed by the terms "*tutorial*" or "*applet*," to obtain a wealth of pedagogical information about a topic not covered in this appendix. Preceding the references we present a handful of URLs that are helpful in providing additional information.

DNA is helix-shaped molecule whose constituents are two parallel strands of *nucleotides*. There are four types of nucleotides in DNA and they correspond to the letters A (for *adenine*), T (*thymine*), C (*cytosine*) and G (*guanine*). DNA is usually represented by *sequences* of these four nucleotides. This assumes that only one strand is considered; the second strand is always derivable from the first by pairing A's with T's and C's with G's and vice-versa. That derivation is called finding the reverse complementary pair of a strand.

Genes are contiguous subparts of single-stranded DNA that are templates for producing *proteins*. Genes can appear in either of the DNAs strands. The set of all genes in a given organism is called the *genome* for that organism. The function of DNA material between genes is largely unknown. Certain intergenic regions of DNA (called *noncoding*) are known to play a major role in cell regulation, the process that controls the production of proteins and their possible interactions with DNA.

Proteins are produced from DNA using three operations or transformations called *transcription*, *splicing*, and *translation*. In humans and higher species (*eukaryotes*) the genes are only a minute part of the total DNA that exists in a cell. For the purposes of this article, chromosomes are compact chains of coiled DNA. In more rudimentary types of cells that do not have a nucleus (*prokaryotes*), the phase of *splicing* does not occur.

DNA is capable of replicating itself. The cell machinery that performs that task is called *DNA-polymerase*. Biologists call the capability of DNA for replication and undergoing the above three (or two) transformations the *central dogma.*

Genes are *transcribed* into *pre-RNA* by a complex ensemble of molecules called *RNA-polymerase*. During transcription the nucleotide T (*thymine*) is substituted by another one designated by the letter U (for *uracil*). Pre-RNA can be represented by alternations of sequence segments called *exons* and *introns*. The exons represent the parts of pre-RNA that will be *expressed*, that is, translated into proteins.

Next comes the operation called *splicing*; an ensemble of proteins called the *spliceosome* performs it. Splicing consists of concatenating the exons and excising the introns to form what is known as *mRNA,* or simply RNA.

The final phase, called *translation,* is essentially a "table look-up" performed by complex molecules called *ribosomes* (an ensemble of RNA and proteins). Translation repeatedly considers a triplet of consecutive nucleotides in RNA and produces one corresponding amino acid. The triplet is called a *codon*. In RNA, there is one special codon called a *start codon* and a few others called the *stop codons*. An *open*

reading frame (ORF) is a sequence of codons starting with a start codon and ending with an end codon. The ORF is thus the sequence of nucleotides that is used by the ribosome to produce the sequence of amino acids that makes up a protein.

There are basically 20 amino acids but, in certain rare situations, others can be added to that list. Since there are 64 different codons and 20 amino acids, the "table look-up" for translating each codon into an amino acid is redundant in the sense that multiple codons can produce the same amino acid. The "table" used by nature to perform translation is called the *genetic code*. Due to the redundancy of the genetic code, certain nucleotide changes in DNA may not alter the resulting protein.

Once a protein is produced, it folds (most of the time) into a unique structure in 3D space.

In the 3D representation of a protein, one can distinguish three different types of components: *α-helices*, *β-sheets* and *coils*. The *secondary* structure of a protein is its sequence of amino acids, annotated to distinguish the boundaries of each component: helices, sheets, and coils. The *tertiary* structure of a protein is its 3D representation.

The *function* of a protein is the way it participates with other proteins and molecules in keeping the cell alive and interacting with its environment. Function is closely related to tertiary structure. In *functional genomics*, one studies the function of all the proteins of a genome. One of the important goals of bioinformatics is to help biologists in deciphering the function of proteins.

ACKNOWLEDGMENTS

The author wishes to express his gratitude to Mark Gerstein, Nathan Goodman and the reviewers who provided many suggestions to improve the original manuscript.

REFERENCES

For interesting graphical gallery of biology consult (downloadable drawings) sponsored by the National Health Museum http://www.accessexcellence.org/AB/GG/.

A recommended glossary of genetic terms http://www.ornl.gov/TechResources/Human_Genome/publicat/primer2001/glossary.html.

NCBI (National Center for Biotechnology Information) http://www.ncbi.nlm.nih.gov.

A summary of interesting sites in bioinformatics is given by the URLs.

On line lectures in bioinformatics—Heidelberg http://www.dkfz-heidelberg.de/tbi/bioinfo/Biol/Intro/.

A special interest group with news and pointers http://www.bioinformatrix.com.

Bioinformatics Bulletin Board http://bioinformatics.org/faq/#education.

Bioinformatics resources http://www.brc.dcs.gla.ac.uk/~actan/resources.html.

Interesting and useful URL's on existing courses.

Jackson's Laboratory Web Page with educational links http://www.jax.org/courses.

Course in bioinformatics (recommended set of slides by R. L. Bernstein) http://www.swbic.org/education/bioinfo/.

Highly recommended texts in molecular cell biology [Alberts et al. 2004; Lodish et al. 2003].

Some texts in computational biology or bioinformatics
[Baldi and Brunak 2002; Baxevanis and Ouellette 1998; Campbell and Heyer 2002; Claverie and Notredame 2003; Durbin et al. 1998; Dwyer 2002; Felsenstein 2003; Gonick and Wheelis 1991; Gusfield 1997; Krane and Raymer 2003; Jones and Pevzner 2004; Mount 2001; Orengo et al. 2003; Pevsner 2003, Pevzner 2000; Setubal and Meidanis 1997; Salzberg et al. 1998; Waterman 1995].

Main Journals in BioInformatics
Bioinformatics, Oxford University Press
IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB).
Journal of Computational Biology, Mary Ann Liebert, Inc, Publishers

Note: Many biology journals publish articles related to bioinformatics, e.g., *Science, Nature, Nucleic Acids Research, Journal of Molecular Biology, Proceedings of the National Academy of Sciences (PNAS), etc. In particular Nucleic Acid Research publishes a compendium of URL's in its yearly January issue.*

Yearly Conferences
RECOMB, *Research in Computational Molecular Biology*
IEEE Computer Society Bioinformatics Conference
PSB Pacific Symposium on Biocomputing
ISMB Intelligent Systems for Molecular Biology

Articles and Books

ABELSON, H., ALLEN, D., COORE, D., HANSON, C., HOMSY, G. KNIGHT, JR., . T. F., NAGPAL, R., RAUCH, E., SUSSMAN, G. J., AND WEISS, R. 1995. Amorphous Computing. *Commun. ACM*.

ALBERTS, B., BRAY, D., JOHNSON, A., LEWIS, J., RAFF, M., ROBERTS, K., AND WALTER, F. 2004. *Essential Cell Biology*, 2nd ed. Garland Publishing.

ASHBURNER, M. AND GOODMAN, N. 1997. Informatics: Genome and genetic databases. *Curr. Op. Gen. Develop. 7,* 750–756.

BALDI, P. AND BRUNAK, S. 2002. *Bioinformatics: The Machine Learning Approach*, MIT Press.

BAR-JOSEPH, Z., GERBER, G., GIFFORD, D., AND JAAKKOLA, T. 2002. A new approach to analyzing gene expression time series data. In *RECOMB The Sixth Annual International Conference on Research in Computational Molecular Biology*.

BAXEVANIS, A., AND OUELLETTE, B. F. F. (EDS.). 1998. *Bioinformatics: A Practical Guide to the Analysis of Genes and Proteins*. Wiley, New York.

BENNETT, C., LI, M., AND MA, B. 2003. Linking chain letters. *Sci. Amer.* (June) 77–81.

BROWN, M. P. S., GRUNDY, W. N., LIN, D., CRISTIANINI, N., SUGNET, C. W., FUREY, T. S., ARES, JR., M. AND HAUSSLER, D. 2000. Knowledge-based analysis of microarray gene expression data using support vector machines. *Proc. Nat. Acad. Sci. 97*, 1, 262–267.

CAMPBELL, A. M. AND HEYER, L. 2002. *Discovering Genomics, Proteomics and BioInformatics*. Benjamin Cummings.

CLAVERIE, J. M. AND NOTREDAME, C. 2003. *Bioinformatics for Dummies*. Wiley, New York.

COHEN, J. 2001. Classification of approaches used to study cell regulation: Search for a unified view using constraints and machine learning. *Electronic Transactions in Artificial Intelligence, Machine Intelligence 18. Linköping Electronic Articles in Computer and Information Science* ISSN 1401-9841, 6(025).

COHEN, J. 2003. Guidelines for establishing undergraduate bioinformatics courses. *J. Sci. Educat. Tech. 12*, 4 (Dec.) 449–456.

DEJONG, H. 2002. Modeling and simulation of genetic regulatory systems: A literature review. *J. Comput. Biol. 9*, 1, 67–103.

DELCHER, A., KASIF, S., FLEISCHMANN, R. D., PETERSON, J., WHITE, O., AND SALZBERG, S. L. 1999. Alignment of whole genomes. *Nucl. Acid Res. 27*, 11, 2369–2376.

DUENWALD, M. 2003. Gene is linked to susceptibility to depression. *The New York Times*, July 18, Sect. A, Page 14, Col. 1.

DURBIN, R., EDDY, S., KROGH, A., AND MITCHISON, G. 1998. *Biological Sequence Analysis*. Cambridge University Press, Cambridge, Mass.

DWYER, R. A. 2002. *Genomic Perl: From Bioinformatics Basics to Working Code*. Cambridge University Press, Cambridge, Mass.

FELSENSTEIN, J. 2003. *Inferring Phylogenies*, Sinauer Associates.

FRIEDMAN, N., LINIAL, M., NACHMAN, I., PEER, D. 2000. Using Bayesian networks to analyze expression data. In *Proceedings RECOMB—Computational Molecular Biology*, pp. 127–135.

GILBERT, D. R., WESTHEAD, D. R., NAGANO, N., AND THORNTON, J. M. 1999. Motif-based searching in TOPS protein topology databases. *Bioinformatics 5*, 4, 317–326. Also see http://www.sander.embl-ebi.ac.uk/tops/.

GONICK, L. AND WHEELIS, M. 1991. *A Cartoon Guide to Genetics*. Harper Perennial.

GOODMAN, N. 2002. Biological data becomes computer literate: new advances in bioinformatics. *Curr. Op. Biotech. 13*, 66–71.

GUSFIELD, D. 1997. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press.

HAND, D. J., MANNILA, H., AND SMYTH, P. 2000. *Principles of Data Mining*. MIT Press, Cambridge, Mass.

HUSON, D. H., REINERT, K., AND MYERS, E. W. 2002. The greedy path-merging algorithm for contig scaffolding. *J. ACM 49*, 5 (Sept.), 603–615.

JAIN, A., MURTY, M., AND FLYNN, P. 1999. Data clustering: A review. *ACM Comput. Surv. 31*, 3, 264–323.

JONES, N. C. AND PEVZNER, P. A. 2004. *An Introduction to Bioinformatics Algorithms*, MIT Press, Cambridge, Mass.

KARP, P. 2001. Pathway databases: A case study in computational symbolic theories. *Science 293*, 2040–2044.

KELLY, H. C. 2003. Terrorism and the biology lab. *New York Times Op-Ed Page*, July 2.

KNUTH, D. E. 1993. *Computer Literacy Bookshops Interview* (Dec.) (Available at http://dmoz.org/Computers/History/Pioneers/Knuth,_Donald/).

KRANE, D. AND RAYMER, M. 2003. *Fundamental Concepts of BioInformatics*. Benjamin Cummings.

KROGH, A. 1998. An introduction to hidden Markov models for biological sequences. In S. L. Salzberg, D. B. Searls, and S. Kasif (eds.), *Computational Methods in Molecular Biology*. Elsevier, Amsterdam, The Netherlands, pp. 45–63.

KUIPERS, B. J. 1994. *Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge*. MIT Press, Cambridge, Mass.

LATHROP, R. H. AND SMITH, T. F. 1996. Global optimum protein threading with gapped alignment and empirical pair potentials. *J. Molec. Biol. 255*, 641–665.

LI, H., HELLING, R., TANG, C., AND WINGREEN, N. 1996. Emergence of preferred structures in a simple model of protein folding. *Science 273*, 666–669.

LIANG, S., FUHRMAN, S., AND SOMOGYI, R. 1998. REVEAL, A general reverse engineering algorithm for inference of genetic network architectures. In *Pacific Symposium on Biocomputing 3*, pp. 18–29.

LODISH, H., BERK, A., MATSUDAIRA, P., KAISER, C. A., KRIEGER, M., SCOTT, M. P., ZIPURSKY, L., AND DARNELL, J. 2003. *Molecular Cell Biology*. W.H. Freeman.

LUSCOMBE, N. M., GREENBAUM, D., AND GERSTEIN, M. 2001. What is bioinformatics? A proposed definition and overview of the field. *Methods Inf. Med. 40*, 346–358 (Also available at http://bioinfo.mbb.yale.edu/papers/).

MILLER, W. 2001. Comparison of genomic DNA sequences: Solved and unsolved problems. *Bioinformatics 17*, 5, 391–397.

MITCHELL, T. 1997. *Machine Learning*, McGraw Hill, New York.

MOUNT, D. W. 2001. *Bioinformatics: Sequence and Genome Analysis*, Cold Spring Harbor Press, Cold Spring Harbor, N.Y.

MYERS, E. 1999. Whole genome DNA-sequencing. *IEEE Computat. Eng. Sci. 3*, 1, 33–43.

ORENGO, C. A., JONES, D. T., AND THORNTON, J. M. 2003. *Bioinformatics: Genes, Proteins and Computers*. BIOS Scientific Publishers, Oxford, England.

PARSONS, R. J., FORREST, S., AND BURKS, C. 1995. Genetic algorithms, operators, and DNA fragment assembly. *Mach. Learn. 21*, 1–2, 11–33. (Also see paper by Parsons in *Computational Methods in Molecular Biology*, S. L. Salzberg, D. B. Searls, and S. Kasif (Eds.). Elsevier, Amsterdam, The Netherlands, 1998.)

PEVSNER, J. 2003. *Bioinformatics and Functional Genomics*. Wiley-Liss.

PEVZNER, P. A. 2000. *Computational Molecular Biology: An Algorithmic Approach*. MIT Press, Cambridge, Mass.

RABINER, L. R. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. IEEE 77*, 2, 257–286.

REGEV, E. AND SHAPIRO, E. 2002. Cellular abstractions: Cells as computation. *Nature 419* (Sept.), 419–443.

RIVAS, E. AND EDDY, S. R. 2000. The language of RNA: A formal grammar that includes pseudo knots. *Bioinformatics 18*, 4, 334–340.

SALZBERG, S. L., SEARLS, D. B., AND KASIF, S., EDS. 1998. *Computational Methods in Molecular Biology*. Elsevier, Amsterdam, The Netherlands.

SCHWARTZ, S., ZHANG, Z., FRAZER, K. A., SMIT, A., RIEMER, C., BOUCK, J., GIBBS, R., HARDISON, R., AND MILLER, W. 2000. PipMaker—A web server for aligning two genomic DNA sequence. *Genome Res. 10*, 4 (Apr.), 577–586.

SEARLS, D. B. 1992. The linguistics of DNA. *Amer. Sci. 80*, 579–591.

SEARLS, D. B. 1998. Grand challenges in computational Biology. In *Computational Methods in Molecular Biology*, S. L. Salzberg, D. B. Searls, and S. Kasif, Eds. Elsevier Amsterdam, The Netherlands.

SEARLS, D. B. 2002. The language of genes. *Nature 420* (November), 211–217.

SETUBAL, J. AND MEIDANIS, J. 1997. *Introduction to Computational Molecular Biology*, PWS Publishing.

THIERRY-MIEG, N. 2000. Protein-protein interaction prediction for C. elegans: In *Knowledge Discovery in Biology, Workshop at the PKDD2000 (Conference on Principles and Practice of Knowledge Discovery in Databases)* (Lyon, France, Sept.).

THOMPSON, J. D., HIGGINS, D. G., AND GIBSON, T. J. 1994. CLUSTAL W: Improving the sensitivity of progressive multiple sequence alignment through sequence weighting, positions-specific gap penalties and weight matrix choice. *Nuc. Acid Res. 22*, 4673–4680.

TOMITA, M., HASHIMOTO, K., TAKAHASHI, K., SHIMIZU, T. S., MATSUZAKI, Y., MIYOSHI, F., SAITO, K., TANIDA, S., YUGI, K., VENTER, J. C., AND HUTCHINSON III, C. A. 1999. E-CELL: Software environment for whole cell simulation. *Bioinformatics 15*, 1, 72–84.

WATERMAN, M. S. 1995. *Introduction to Computational Biology: Maps, Sequences and Genomes*. CRC Press.

WATSON, J. D. AND BERRY, A. 2003. *DNA: The Secret of Life*. Knopf.

WETHERELL, C. S. 1980. Probabilistic languages: A review and some open questions. *ACM Comput. Surv. 12*, 4, 361–379.

ZUKER, M. AND STIEGLER, P. 1981. Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. *Nuc. Acids Res. 9*, 133–148. (Also see http://www.bioinfo.rpi.edu/~zukerm/).

Received July 2003; accepted August 2004