REPRINT: originally published as: Robbins, R. J., 1992. Challenges in the human genome project. *IEEE Engineering in Biology and Medicine*, (March 1992):25–34.

CHALLENGES IN THE HUMAN GENOME PROJECT

PROGRESS HINGES ON RESOLVING DATABASE AND COMPUTATIONAL FACTORS

Robert J. Robbins

Applied Research Laboratory
William H. Welch Medical Library
The Johns Hopkins University

TABLE OF CONTENTS

REPRINT: originally published as: Robbins, R. J., 1992. Challenges in the human genome project. *IEEE Engineering in Biology and Medicine*, (March 1992):25–34.

CHALLENGES IN THE HUMAN GENOME PROJECT

Robert J. Robbins

Although the Human Genome Project is well recognized as the first Big Science project in biology, it is less well known as a major project in computer technology and information management. By the time this project is finished, many of its innovative laboratory methods will have begun to fade from memory. Although a few might be preserved as exercises for undergraduates, the rest will soon become footnotes in the history of molecular techniques. What will remain, as the project's enduring contribution, is a vast body of computerized knowledge. Seen in this light, the Human Genome Project is nothing but the creation of the most amazing database ever attempted—the database containing instructions for building people.

The 3.3 billion nucleotides in the DNA of a human gamete constitute a single set of these instructions. With each nucleotide represented as a single letter, one copy of this sequence, typed (in standard pica typeface) on a continuous ribbon of material, could be stretched from San Francisco to New York and then on to Mexico City. No unaided human mind could hope to comprehend such a mass of information. Just assembling, storing, publishing, and distributing (much less understanding) such a sequence will require automation. Representing individual variations and managing a fully annotated, functionally described version of the sequence is probably beyond current information–handling technology.

Even now, when the Human Genome Project is merely in the first year of its first five–year plan, computer systems are playing an essential role in all phases of the work. Laboratory databases help manage research materials while computer–controlled robots perform experimental manipulations. Automated data–acquisition systems log experimental results and analytical software assists in their interpretation. Local database systems store the accumulating knowledge of a research team, while public databases provide a new kind of publication for scientists to share their findings with the world.

Some genomic database and software problems are fairly straightforward. Others will push the envelop of information–management theory. The HGP requires a continuum of database activities, ranging from application development to research. The research community needs production–quality, rock–solid, public–access databases right now, but pure computational research will be required to develop the new ideas and technologies necessary for the production–quality databases of a decade hence. The challenges of the Human Genome Project

will drive computational science, just as earlier challenges from genetics drove the development of modern statistical analysis.

In the Human Genome Project, computers will not merely serve as tools for cataloging existing knowledge. Rather, they will serve as instruments, helping to create new knowledge by changing the way we see the biological world. Computers will allow us to see genomes, just as radio telescopes let us see quasars and electron microscopes let us see viruses.

PURPOSE AND SCOPE

The Human Genome Project (HGP) is an international undertaking with the goal of obtaining a fully connected genetic and physical map of the human chromosomes and a complete copy of the nucleotide sequence of human DNA. As such, it has been described as the first "big science" project in biology [3], [15]. Although the computational challenges associated with the project have been described [12], [14], some computer scientists have expressed concerns about its complexity: "Computationally, the project is trivial. The human genome is nothing but a string of 3.3 billion characters. Where is the challenge in representing or manipulating this? Biologists may think that 3.3 gigabytes is a lot of data, but a database of this size is routine in many application areas."

Such skeptics are simply wrong. The HGP can be logically divided into two components, getting the sequence and understanding the sequence, but neither involves a simple 3.3 gigabyte database with straightforward computational requirements. A computer metaphor can help establish the scope of the effort. Consider the 3.3 gigabytes of a human genome as equivalent to 3.3 gigabytes of files on the mass–storage device of some computer system of unknown design. Obtaining the sequence is equivalent to obtaining an image of the contents of that mass–storage device. Understanding the sequence is equivalent to reverse engineering that unknown computer system (both the hardware and the 3.3 gigabytes of software) all the way back to a full set of design and maintenance specifications.

Securing the sequence is further complicated by the fact that the mass–storage device is of unknown design and cannot simply be read. At best, experimental methods can be used to obtain tiny fragments of sequence from it. Because these experimental methods are expensive ($5–10 per byte) and error prone, new techniques are constantly being developed and tested. Meanwhile, a database must be designed to hold the fragments that are obtained, along with a full description of the procedures used to generate them. Because the experimental procedures change rapidly and often radically, this is equivalent to designing and maintaining a database for an enterprise whose operating procedures and general business rules change weekly, perhaps daily, with each change requiring modifications to the database schema.

As the sequence fragments accumulate, efforts will be made to synthesize the fragments into larger images of contiguous regions of the mass–storage device, and these synthesized fragments must also be represented in the database. If multiple inferences are consistent with the present data, all of the consistent possibilities must be represented in the database to serve as the basis for further reasoning when more data becomes available. As even larger regions are synthesized, the entire cascade of premisses, procedures, and logical dependencies must be stored, so that a "logical roll–back" can occur if some new observation renders an earlier premise doubtful. Since all of the data are obtained experimentally, each observation and deduction will have some error term associated with it. The deductive procedures must use these errors to assign probabilities to the various deductive outcomes obtained and stored in the database. Thousands of researchers from independent laboratories will be using the system and contributing data. Each will have a notion of proper error definition and of the rules by which the errors should be combined to provide reliability estimates for the composite sequences. Therefore, the database must be capable of supporting probabilistic views and of returning multiple answers, each with its own associated conditional probabilities, to any query. The involvement of many independent researchers, each employing slightly different experimental procedures and concepts, will also result in extensive nomenclatural synonymy and homonymy. The database must take these nomenclatural difficulties into account and should be able to present users with views consistent with their own preferred usage.

Reverse engineering the sequence is complicated by the fact that the resulting image of the mass–storage device will not be a file–by–file copy, but rather a streaming dump of the bytes in the order they occupied on the device and the files are known to be fragmented. In addition, some of the device is known to contain erased files or other garbage. Once the garbage has been recognized and discarded and the fragmented files reassembled, the reverse engineering of the codes must be undertaken with only a partial, and sometimes incorrect understanding of the CPU on which the codes run. In fact, deducing the structure and function of the CPU is part of the project, since some of the 3.3 billion gigabytes are known to be the binary specifications for the computer–assisted–manufacturing process that fabricates the CPU. In addition, one must also consider that the huge database also contains code generated from the result of literally millions of maintenance revisions performed by the worst possible set of kludge–using, spaghetti–coding, opportunistic hackers who delight in clever tricks like writing self–modifying code and relying upon undocumented system quirks.

One computer scientist, upon hearing this metaphoric description, opined that, far from being trivial, the HGP was simply impossible: "Why, that's like working with both hands tied behind your back, blindfolded, in a vacuum!" The Human Genome Project isn't impossible, but it is complex. The goal of this paper is to provide an overview of the HGP that emphasizes its generic problems and computational challenges. Presentations of actual current database and computational efforts are available elsewhere [1], [2], [4], [5], [6], [10], [11].

BASIC BIOLOGICAL CONCEPTS

Early on, biochemists established that an individual's biological structure and function are controlled by proteins—a workhorse set of molecules that occur in thousands of forms to perform thousands of functions. Since the attributes of an organism are ultimately determined by the types and quantities of proteins present in its cells, these molecules are clearly the fundamental building blocks of life. The human body contains 50,000 to 100,000 different kinds of protein.

Although it is their three–dimensional configuration that gives proteins their functional specificity, chemically proteins are linear polymers called polypeptides that contain hundreds of amino acids, linked by peptide bonds into a continuous sequence. Proteins assume their three–dimensional shape "automatically" once they are synthesized with a specific sequence of amino–acid subunits. Although many kinds of amino acids exist, only twenty different forms are used in proteins. Because proteins are linear polymers containing just twenty different subunits, the structure of any given protein molecule can be specified with a linear string using a twenty–letter alphabet.

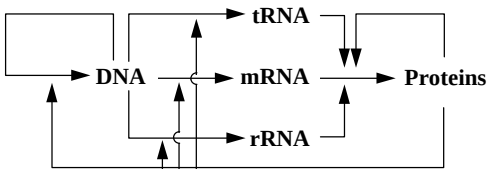
In the first half of this century, classical geneticists showed that the control of biological structure and function is passed from generation to generation in the form of genes—hypothetical entities that occur singly in gametes (sperm and eggs) and doubly in organisms. By 1950, it was apparent that genes had to act by controlling the synthesis of proteins, but the means by which this might be accomplished were a mystery.

With the demonstration that genes are made of deoxyribo–nucleic acid (DNA) and the discovery of the structure of DNA, the science of molecular biology was established and the first possibility of understanding gene function appeared. DNA was found to be a linear polymer of molecular subunits called nucleotides, which occur in DNA in four specific forms: adenine, thymine, cytosine, and guanine (usually abbreviated as A, T, C, G). Because DNA is a linear polymer containing just four different subunits, the structure of any given DNA molecule can be fully specified with a linear string using a four–letter alphabet.

Since DNA and proteins can both be specified as linear strings, researchers quickly hypothesized that genes might control the synthesis of proteins by simply encoding their amino–acid sequences as nucleotide sequences. This proved to be true, with the proviso that instructions encoded in DNA are first transcribed into ribonucleic acid (RNA) polymers before being translated into the amino–acid sequence of proteins. (RNA differs from DNA by carrying an extra hydroxyl group on each nucleotide, and by carrying the nucleotide uracil, abbreviated U, wherever DNA carries thymine.) When this hypothesis was established, the "fundamental dogma" of molecular biology was born: DNA directs the synthesis of RNA, which directs the synthesis of protein, often illustrated as

DNA ⟶ RNA ⟶ Proteins

The production of RNA from a DNA template is called transcription and the production of protein from an RNA template is known as translation. Complexity was added to the fundamental dogma with the recognition that (1) DNA redundantly encodes for its own duplication, (2) DNA–directed protein synthesis involves three different classes of RNA (tRNA, mRNA, and rRNA), and (3) previously synthesized proteins in the form of enzymes are also key actors in both DNA replication and protein synthesis:



The specific instructions coding the amino–acid sequence for a particular protein are carried in the nucleotide sequence of a particular mRNA, which is transcribed from a particular gene in DNA. The means (Figure 1) by which mRNA sequences determine amino–acid sequences has proven to be the same for all living things on this planet.

5´	U	C	A	G	3´
U	phe phe leu leu	ser ser ser ser	tyr tyr STOP STOP	cys cys STOP trp	U C A G
C	leu leu leu leu	pro pro pro pro	his his gln gln	arg arg arg arg	U C A G
A	ile ile ile met	thr thr thr thr	asn asn lys lys	ser ser arg arg	U C A G
G	val val val val	ala ala ala ala	asp asp glu glu	gly gly gly gly	U C A G

Figure 1. The universal code by which genetic information in mRNA is translated into protein. The nucleotides in an mRNA molecule are read in non–overlapping groups of three, called codons. The “reading frame” is established by requiring that each protein always begin with the codon AUG. As each successive codon is encountered, the protein–synthesis machinery incorporates the amino acid given in this table. When a stop codon (UAA, UAG, or UGA) appears, synthesis is complete. This code is used by all living things

on this planet. The twenty amino acids commonly found in proteins, and their three–letter abbreviations are:

ala	alanine	leu	leucine
arg	arginine	lys	lysine
asn	asparagine	met	methionine
asp	aspartic acid	phe	phenylalanine
cys	cysteine	pro	proline
gln	glutamine	ser	serine
glu	glutamic acid	thr	threonine
gly	glycine	trp	tryptophan
his	histidine	tyr	tyrosine
ile	isoleucine	val	valine

Life is fundamentally digital, not analog; genetic information is passed from generation to generation in the form of a discrete code. The parallel analogy with the digital encoding found on the mass–storage devices of computer systems is almost inescapable. Capturing and understanding all of the encoded information in human genes is the long–term goal of the human genome project.

Although the mRNA–to–protein code is straightforward, the actual process by which information stored in a human gene becomes transformed into a protein is considerably more complex, as shown schematically in Figure 2.

If we think of information encoded in genes as equivalent to programs encoded on a mass–storage device, and the biological functions performed by proteins as the execution of these programs, then the steps in Figure 2 that are labelled “post–transcriptional processing” and “post–translational processing” represent the actions of self–modifying code, since they involve changes to encoded instructions performed after the instructions are “loaded” but before they “execute.” Worse than simple self–modifying code, the protein enzymes that carry out this post processing are more similar to software daemons that run constantly, activating only when a particular program is loaded and then modifying that program’s code in memory before it starts executing. Reverse engineering self–modifying code is notorious difficult.

Because previously synthesized proteins effect, affect, and control all aspects of the expression of genetic information, reverse engineering the human genome will be complex, since these protein daemons interact with control signals carried in DNA to regulate the expression of genes differently in different tissues. The human genetic apparatus is not a mere collection of recipes for building proteins, for if it were, cells carrying the same set of genes could not differentiate into a variety of tissues, such as brain and muscle. Although many control mechanisms are known, the majority are not yet well understood. Identifying and deciphering them is a major goal of genomics.

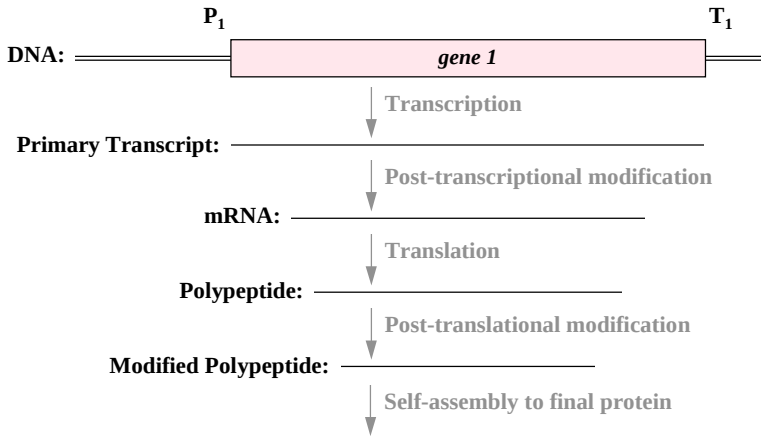


Figure 2. A schematic illustration of the steps involved in DNA–directed protein synthesis. Start and stop signals encoded in the DNA tell enzymes to begin and end transcription, producing a primary RNA transcript. Other enzymes modify the transcript by adding or deleting nucleotides. Most human genes contain large regions of non–coding sequences (introns) interspersed with coding sequences (exons). The intron sequences are removed during post–transcriptional processing, yielding a final mRNA. The mRNA is translated into protein, using the universal code along with start and stop signals embedded in the mRNA before and after the coding region. Translation produces a polypeptide that itself may be subject to significant enzymatic modification. The resulting polypeptide finally assumes its three–dimensional shape and becomes a functional protein.

UNDERSTANDING THE SEQUENCE

Soon after obtaining a DNA sequence, researchers try to identify and understand its function through a mixture of logic and experimentation. Attempting to understand a sequence of hexadecimal values from the mass–storage device of some computer system would involve similar steps. This section compares the two processes.

Direct Sequence Interpretation

Understanding an arbitrary sequence is much easier if the system and context in which the sequence is to be interpreted are specified. For example, consider the following RNA sequence:

```
AUG GUG CAC CUG ACU CCU GAG GAG AAG UCU GCC GUU
```

If it can be assumed that this the beginning of a coding region on an mRNA molecule (this is, in fact, the beginning of the coding region in human ß–hemoglobin mRNA), then interpreting the sequence requires a trivial look up in the mRNA–to–protein dictionary, yielding:

```
AUG GUG CAC CUG ACU CCU GAG GAG AAG UCU GCC GUU ···
met val his leu thr pro glu glu lys ser ala val ··
```

Determining the actual function of a particular protein (which depends upon its three–dimension structure) from its sequence, however, is not straightforward and in fact is impossible with current technology. Until better algorithms are developed, the possible outcome space for such an n–body problem is too large to compute in reasonable time.

Now, consider the following hexadecimal sequence:

```
CD 05 CD 20
```

If it can be assumed that this is executable code from an Intel–based, MS–DOS computer system, a knowledge of the op codes would permit the reverse assembly to:

```
CD05  INT 05
CD20  INT 20
```

Here, too, determining actual function from an interpretation is not entirely straightforward. For example, in MS–DOS, calling interrupt 5 triggers the “print screen” routine in BIOS and interrupt 20 is the old “program terminate” (equivalent to the CP/M BDOS function 00H) op code carried over from the earliest versions of MS–DOS. A first hypothesis, then, would be that “CD 05 CD 20” could function as a self–contained executable program that printed whatever happened to be on the screen, then returned control to the operating system.

This hypothesis, however, is not necessarily correct. Calling DOS interrupt 5 passes control to whatever routine is pointed to by the low–memory vector for INT 5. Although normally the “print screen” routine, it could be any routine left in memory by a previously executed terminate–and–stay resident program (c.f., [9], p 128). Determining what this four–byte program actually would do at a given time in a particular machine would involve some combination of experimentation and further analysis of the interrupt vector table and the code addressed by the INT 5 vector.

Determining the function of a DNA sequence involves similar steps. After obtaining a context in which to interpret the sequence (so that the relevant “op codes” are known) analysis begins. Because the full set of biological op codes is not yet known, and because biological subsystems are so interdependent, considerable experimentation and comparative work is required for researchers to generate a tentative understanding of the sequence. Part of the database and computation challenges of the HGP is building databases for storing evolving hypotheses regarding these biological op codes, with the ultimate goal of using these data to write a “biological disassembler” that could recognize and interpret all functional regions in any arbitrary piece of DNA.

Comparative Sequence Analysis

The previous discussion examined the interpretation of arbitrary sequences when something is known of the system and context in which they operate. But what if the system were unknown and the context uncertain? To illustrate one approach to interpreting sequences from an unknown computer system, let us first assume that we know nothing of Intel–based computers or of MS–DOS or of ASCII, then consider the following examples.

When invoked, a particular MS–DOS program named WARMBOOT.COM causes the same effect as pressing the ctrl–alt–delete keys—that is, it causes the system to reboot without performing any of the system checks associated with a cold boot. This program, in its entirety, is found to consist of the sequence:

```
BA 40 00 8E DA BB 72 00 C7 07 00 00 EA 00 00 FF FF
```

How might this program work and the sequence be interpreted? If one truly knew nothing about this computer system, very little could be done with just this sequence. But, suppose that another program named COLDBOOT.COM (that does what the name implies) were also known and that the program, in its entirety, consisted of the sequence:

```
BA 40 00 8E DA BB 72 00 C7 07 34 12 EA 00 00 FF FF
```

By aligning the two sequences and associating differences in their structure with differences in their function, a beginning, however feeble, toward reverse engineering MS–DOS machine code could be generated.

```
WARMBOOT: BA 40 00 8E DA BB 72 00 C7 07 00 00 EA 00 00 FF FF
COLDBOOT: BA 40 00 8E DA BB 72 00 C7 07 34 12 EA 00 00 FF FF
```

Substituting “34 12” for “00 00” somehow changes the WARMBOOT program into a COLDBOOT program. These bytes must be data, and the remainder instructions for invoking the boot routine. This trivial comparison has allowed us to get a small purchase on the problem.

Now consider a set of programs known to have similar functions. Program 1 displays “Hello world” on the terminal, and programs 2, 3, and 4 display “Hi world,” “Goodbye world,” and “Hello,” respectively. If these are placed into a multiple alignment (which requires inserting gaps to bring the similar regions into apposition), we have:

```
1: EB 0D 90 48 65 6C 6C 6F -- -- 20 77 6F 72 6C 64 24 B4 00 B4 09 BA 03 01 CD 21 C3
2: EB 0A 90 48 69 -- -- -- -- -- 20 77 6F 72 6C 64 24 B4 00 B4 09 BA 03 01 CD 21 C3
3: EB 0F 90 47 6F 6F 64 62 79 65 20 77 6F 72 6C 64 24 B4 00 B4 09 BA 03 01 CD 21 C3
4: EB 07 90 48 65 6C 6C 6F -- -- -- -- -- -- -- -- 24 B4 00 B4 09 BA 03 01 CD 21 C3
```

Even with no prior knowledge of the op codes or the character codes used by MS–DOS systems, useful insights could be obtained from these alignments and a knowledge of the four programs' functions. The code sequences clearly have regions of identity and regions of variability, and a reasonable first hypothesis would be that the variable regions contain codes for the characters to be displayed, and the constant regions contain instructions for displaying these characters.

An analysis of the variable regions could lead to a tentative deduction of the entire character code, since the variable regions each seem to contain exactly the same number of codes as of characters displayed, and adjacent letters in the alphabet apparently have sequential code values. The hypothesized complete set of character codes could be tested by replacing bytes in the variable region with other values, then executing the program and recording what characters are displayed. In fact, substituting all values from 00 through FF would allow the rapid determination of the entire character code.

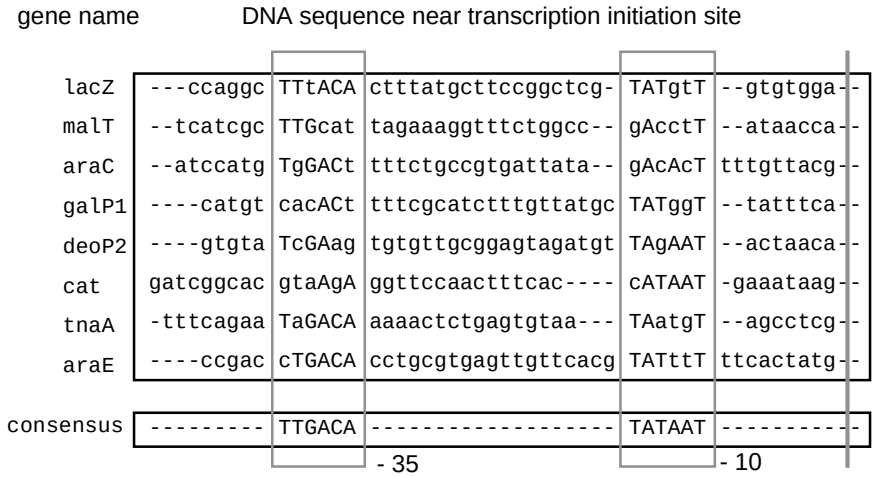
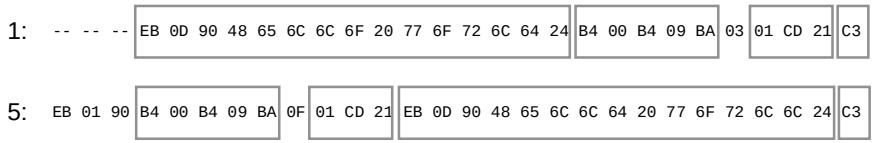


Figure 3. An excerpt from an alignment performed in an effort to understand the DNA control region that signals START OF TRANSCRIPTION for genes of the bacterium *Escherichia coli* [8]. The last nucleotide in each sequence (bracketed with arrowheads) are known to be the point of transcription initiation, with transcription proceeding to the right. Analysis of these and many other sequences has found two regions of similarity upstream from the start of transcription, one approximately 10 bases upstream (the –10 consensus sequence) and the other approximately 35 bases upstream (the –35 consensus sequence). Averages of base occurrences taken over many sequences have shown these two upstream regions to be characterized by the consensus sequences as given in the figure. Interestingly, even though the contents of the variable region between the consensus sequences seems to have little or no effect upon the efficiency of the signal in initiating transcription, the length of the variable regions does seem to have an effect, thereby demonstrating some of the subtlety involved in biological coding. Directed mutagenesis studies have shown that

changes toward or away from the consensus sequences increase or decrease, respectively, the ability of a DNA region to act as a site of transcription initiation.

Similar alignments of molecular sequences are commonly employed in molecular biology. And, making specific substitutions in DNA, called directed mutagenesis, is also an important experimental technique for studying biological function and for deciphering the biological op codes in DNA. As an early example, Hawley et al. [8] compared regions in bacterial DNA known to be the site of transcription initiation. As Figure 3 shows, this alignment did detect regions of similarity. However, as the figure also shows, "genomic computers" are inherently probabilistic. That is, with the exception of the mRNA–to–protein codes, most biological op codes occur in a variety of forms, with the different forms variously affecting the *probability* that a particular event will occur. Reverse engineering a system with probabilistic codes will certainly be more challenging than would be the case for a system employing deterministic codes.

Returning to our computer example, let us consider another alignment, this time between two programs with identical functionality: both write "Hello world".

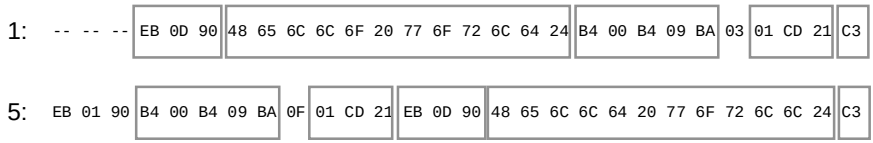


This "alignment" is more complex, but it also is especially informative when compared with the analysis of the previous multiple alignment. Sequences 1 and 5 contain four identical subregions, but with three of them in a different order. If the order of blocks is ignored, these two sequences are nearly identical. Twenty four of the twenty five bytes in string 1 have an exact match in string 2.

Likewise, proteins and genes may contain permutable functional blocks. Developing string–matching algorithms and writing software to produce "non–linear alignments" (i.e., the recognition of variously ordered similar subregions), even when the sequences involved may contain hundreds of thousands of characters, is one computational challenge of the HGP. Another challenge is developing an indexing method for some kind of n–dimensional "similarity space" so that queries like "SELECT ALL SEQUENCES WHERE SIMILAR TO SEQUENCE X" will execute in reasonable time, even if run against a database containing millions of sequences comprising more than a terabyte of data.

At present, whenever a new DNA sequence is added to a DNA database, a brute–force similarity comparison is made between that sequence and every other sequence in the database. This is resource–intensive work, and it has been estimated that within ten years the databases will have grown and the rate of sequence acquisition will have increased so that it would require a teraflop machine running all out, twenty four hours a day, just to accession and catalog incoming sequence data. However, an appropriate similarity–space index could entirely eliminate this computational burden.

Returning again to our hexadecimal alignment problem, from the previous multiple alignment of sequences 1 through 4 it was possible to hypothesize that the common region of code that began with “24” and ended with “C3” was an invariant block that contained instructions for writing a message to the screen. However, from the alignment of sequences 1 and 5 it now appears that “24” is more likely associated with the end of the variable block containing the message and that “C3” more likely denotes the end of the entire program rather than just the end of the display–message block. There is also a sequence that is nearly equivalent to the invariant, display–message block, but with “0F” substituted for “03.” Furthermore, we see that the two large identical blocks between sequences 1 and 5 that begin with “EB 0D 90” should probably be decomposed into two blocks, as:



The blocks of identical or similar code can now be categorized as data (containing characters to be displayed) or as instructions. Previous work deciphering the character code should have established that “24” represents “$,” and our current analysis has observed that all of the variable strings end with this symbol, even though it is not displayed. Is “$” used as punctuation to terminate strings? The “C3” code seems to be the terminate–program code, and “EB” is always followed by a hexadecimal digit giving the distance in bytes to the beginning of the next executable block. Perhaps “EB” is the jump instruction. The “90” code seems to be doing nothing. Could it be a NO OP? Comparing the invariant instruction block of the first multiple alignment with the equivalent blocks in this last analysis, we see that something changes in the middle of the coding block. Since this seems to be associated with a change in the relative position of the variable character block, perhaps it represents an address for the character block.

Using such techniques, it would be possible in theory to reverse engineer first the entire set of Intel op codes and then all application codes that run on such machines. The data– and hypothesis–management requirements for such an effort would be daunting. We would somehow have to record, say, not only that our current hypothesis is that “C3” is the program–terminate code, but we would also be required to track of all of the evidence and reasoning to support that hypothesis. At present, we would need to record that “03” and “0F” seem to involve addressing and we would need some way to modify that when additional information is obtained. We would have to store all known program sequences in a database, linked to the analyses that had been done upon them. We would need to retrieve sequences according to their similarity. Of the five screen–writing sequences just analyzed, numbers 1 and 5 are functionally the most similar, but no simple indexing scheme based upon their linear contents would ever place them

next to each other. Although we could "eyeball" the alignment of short sequences, software would need to be developed to help us perform optimum alignments for large sequences.

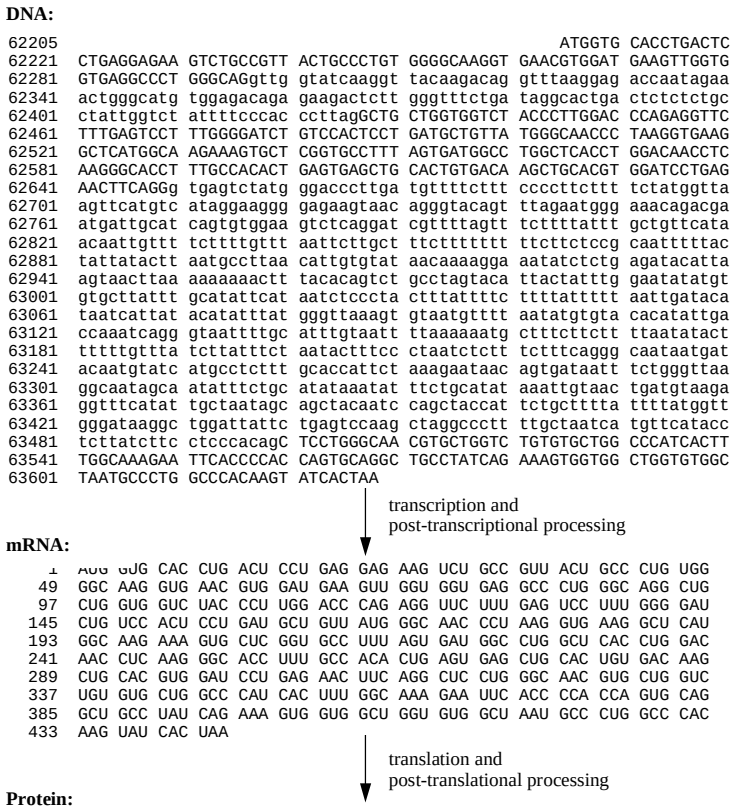


Figure 4. An illustration of the decoding involved in the synthesis of human ß–hemoglobin from information encoded in the DNA of human chromosome 11. The caption "DNA" labels an excerpt containing the coding region of the gene for ß–hemoglobin, taken from a 73,326–base sequence (Genbank) spanning a region on the short arm of chromosome 11. To produce the actual protein, all of the DNA is first copied into RNA (in which all of the T's are replaced with U's). Next, the RNA is subjected to post–transcriptional processing that removes all of the bases transcribed from those shown in lower case. The resulting functional mRNA is then translated into protein according to the universal code shown in

Figure 1. Finally, the first amino–acid (methionine) is removed during post–translational processing, yielding a molecule of normal human ß–hemoglobin containing 146 amino acids arranged in a specific sequence.

If we were to reverse engineer a full 3.3 gigabytes of files, while at the same time deducing the op codes and architecture of the CPU, the database requirements for recording of all of our experimental observations and tentative hypotheses would be enormously complex. If the 3.3 gigabytes of files were written by undisciplined hackers prone to clever tricks, our work would be rendered much more difficult. So it is with the reverse engineering of genomes. Gigabytes of sequence, once obtained, are just the beginning. The database requirements are horrendously complex, because even the concepts and definitions of the objects to be databased can change with each new observation.

Improved methods need to be developed for automatically aligning, interpreting, and decoding biological sequences. Despite the universality of the mRNA–to–protein dictionary, the presence of complex, not fully understood, control sequences within coding regions makes even automatic decoding still an unsolved problem. Figure 4 shows an actual DNA sequence that codes for ß–hemoglobin, a component of the functional hemoglobin that carries oxygen in the blood.

Notice that there are non–coding regions (called introns) interspersed within the coding regions. Even individual codons may be divided. Before the RNA transcript from this gene becomes functional mRNA, the introns must be removed, or "spliced out." Although researchers have empirically determined the point of splicing ("splice junctions") for this gene, the algorithmic detection of all splice junctions in all genes cannot be done. We have not yet determined precisely how splice junctions are encoded in DNA, despite the fact that alignment analysis has detected some apparent consensus sequences. Thus, to allow the useful deduction of protein sequences from DNA sequences in a genome database, the DNA sequences must be accompanied with a significant amount of annotation, much of which must be determined empirically and entered by hand.

Because human–genome researchers are interested in the biological and medical effects of human genes, descriptions and commentaries regarding these must also be collected and stored in databases. The human ß–hemoglobin gene spans a mere 2000 nucleotides, yet the current commentaries about it in various databases (e.g., GenBank, for annotated DNA sequences, PIR, for annotated protein sequences, and OMIM, for medical commentary on human genes) collectively already contain more than 500,000 bytes of information. If such an information amplification occurred over the entire genome, the HGP would ultimately involve terabytes of processed information and commentary.

Although this amplification ratio is not likely apply evenly to all portions of the genome, it might well apply to all genes. The ß–hemoglobin gene just happens to be one of the best studied human genes to date. As the Human Genome Project and related research continue, the data– and information–handling problems associated with the understanding–the–sequence component of the project will

certainly challenge, in complexity and in volume, the capabilities of database technology.

OBTAINING THE SEQUENCE

DNA in cells is organized into structures called chromosomes, each of which consists of one long DNA molecule, accompanied by numerous protein molecules. Normal human cells carry 46 chromosomes; 23 are contributed by each parent. Although human cells are not visible to the naked eye, they contain DNA molecules which, stretched and layed end to end, would be more than three feet long. Human chromosomes occur in a variety of sizes over approximately a 5:1 ratio (Figure 5).

chromosome	percent of genome	number of base pairs
1	8.10%	267,142,857
2	7.71%	254,571,429
3	6.48%	213,714,286
4	6.00%	198,000,000
5	5.81%	191,714,286
6	5.62%	185,428,571
7	5.14%	169,714,286
8	4.67%	154,000,000
9	4.57%	150,857,143
10	4.38%	144,571,429
11	4.38%	144,571,429
12	4.38%	144,571,429
13	3.62%	119,428,571
14	3.43%	113,142,857
15	3.24%	106,857,143
16	3.14%	103,714,286
17	3.05%	100,571,429
18	2.76%	91,142,857
19	2.57%	84,857,143
20	2.38%	78,571,429
21	1.81%	59,714,286
22	1.90%	62,857,143
X	4.86%	160,285,714
Y	2.10%	69,142,857
Total in Genome:		3,300,000,000
Average per chromosome:		143,478,261

Figure 5. The relative sizes and estimated nucleotide content of the 22 human autosomes and the X and Y sex chromosomes.

Because chromosomes are molecules only a few atoms wide but several inches long, they are fragile and break easily if manipulated. In the course of their work,

molecular biologists break them into random fragments. These fragments are then picked up individually by thousands of "vector" micro–organisms, each of which carries a human fragment and replicates the human DNA along with its own DNA. When a large population is later grown from a single individual micro–organism that is carrying just one fragment from human DNA, that population provides a ready source of multiple copies of a particular small region of human DNA. Complete sets of such clones, each carrying different fragments from the entire genome are known as "libraries."

Although cloning techniques provide ready sources of human DNA, they provide no immediate way to determine the precise location in the human chromosomes from which the DNA originated. Determining this location requires further experimentation. Because the fragments are generated at random, a library set of fragments spanning a total length far in excess of one human genome is required in order to ensure a reasonable probability that any particular piece of human DNA will be carried in at least one clone. The ratio of excess DNA that must be cloned in order to ensure reasonable coverage of the genome is, of course, a function of the size of the fragments generated and of the degree of certainty required. With modern clone–manipulation technology, about a five–fold set of DNA must be cloned to generate adequate coverage of the genome. This means that a good human genome clone library would contain between 15 and 20 billion base pairs of human DNA.

If each of these clonal fragments could be readily sequenced (i.e., nucleotide sequence determined), assembling the final human genome would be straightforward. The clonal sequences would be compared with each other, regions of overlap detected, and the final sequence assembled. However, at present, sequencing DNA is expensive (about $5–10 per base) and time consuming. To avoid the waste that completely sequencing a five–fold redundant set of DNA would entail, one of the preliminary goals is to study the clones in an effort to determine the minimum spanning set of DNA fragments required to cover the entire genome. Then, when sequencing techniques have been improved to a cost of less than $0.50 per base (another of the goals of the HGP), an all–out effort to sequence the minimal spanning set will commence.

There are many techniques for building minimal spanning sets, and improved ones are constantly being developed. Basically, each technique involves performing some partial characterization on each fragment and then comparing the partial characterization scores for each pair of fragments to determine the probability of overlap between the two fragments. The resulting NxN probability matrix is used to deduce sets of overlapping fragments. Each set of contiguous, overlapping fragments is known as a contig. Once the entire sequence is spanned by one large contig, the resulting minimal set of spanning fragments can be sequenced and the final, overall sequence assembled.

The process of fragment characterization and contig assembly is complicated by the occurrence of both random and systematic error. Some of the partial characterization measurements may be in error, some clones may actually carry

fragments from two or more locations in the genome, some different regions of the genome may carry identical sequences, some specific human sequences may be systematically resistant to being incorporated in micro–organism clones, and finally, some additional sources of error are undoubtedly as yet unknown.

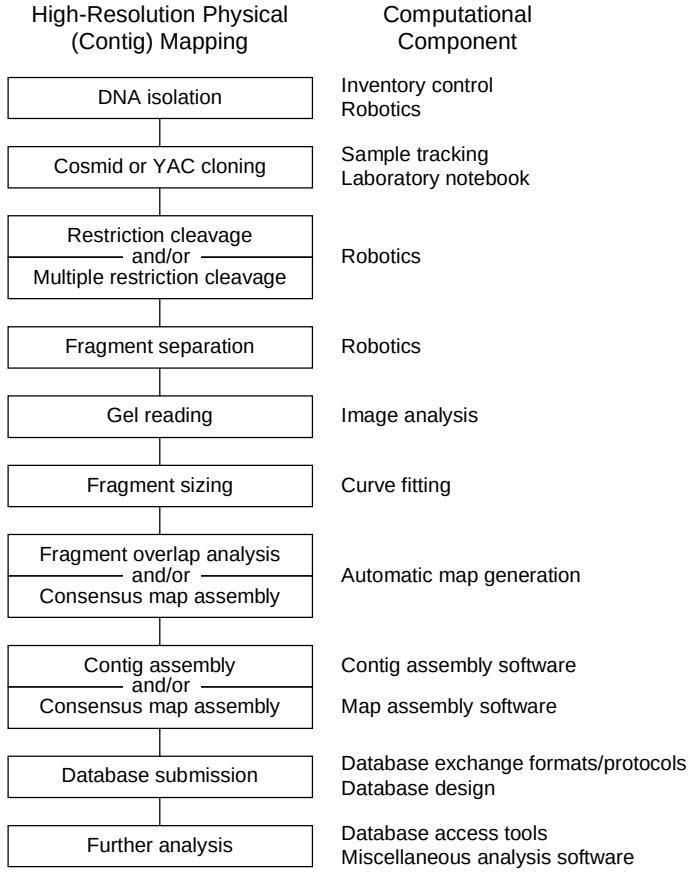


Figure 6. Schematic diagram of the experimental steps involved in high–resolution physical mapping of chromosomes. The column on the right gives the computational activities associated with each experimental step. (Figure adapted from [12].)

The information–handling requirements for this work are: build a database that can (1) hold all of the different and inconsistent and rapidly changing data and metadata describing the sequence fragments and their partial characterizations as they are obtained; (2) track the assembly of fragments into provisional contigs; (3) represent error and uncertainly associated with nearly every measurement and inference; (4) rapidly adapt to recording data for experimental procedures that may

change almost daily; and (5) allow the comparison and merger of contigs prepared with different experimental and computation techniques.

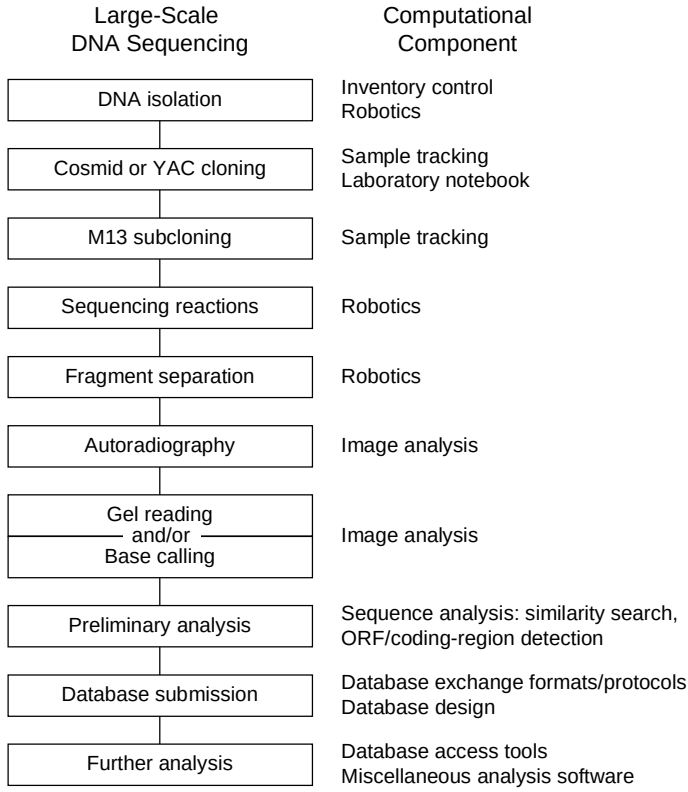


Figure 7. Schematic diagram of the experimental steps involved in large–scale DNA sequencing. The column on the right gives the computational activities associated with each experimental step. (Figure adapted from [12].)

It is hoped that ultimately the various efforts of multiple researchers will converge upon a single, correct set of contigs that span the entire sequence. However, while the work is ongoing, it will be necessary to maintain in the database all of the different, inconsistent, and overlapping versions for subparts of the problem.

Every stage in contig assembly and in bulk sequencing benefits from computer assistance. Since actual experimental manipulations involve handling tens of thousands of tiny samples, robotics are required to keep errors at a minimum. Raw data from these experiments come in the form of images, whose manual translation into numerical form is prohibitively time consuming. Because valid biological experiments must be capable of replication, computerized inventory control is

required to track the literally tens of thousands of components involved in complex experimental designs. As data are analyzed, determining the optimum next experiment may involve complex combinatorics, so laboratory–assistant software is needed to plan and manage laboratory experimentation. Figures 6 and 7 show some of the experimental steps, and their computational counterparts, for both contig assembly and large–scale DNA sequencing.

Ullman [13] has commented on current database systems:

> The modification of the database scheme is very infrequent, compared to the rate at which queries and other data manipulations are performed... The classical form of database system ... was designed to handle an important, but limited class of applications. [For example,] files of employees or corporate data in general, airline reservations, and financial records. The common characteristic of such applications is that they have large amounts of data, but the operations to be performed on the data are simple. In such database systems, insertion, deletion, and retrieval of specified records predominates, and the navigation among a small number of relations or files ... is one of the more complex things the system is expected to do.

With genome laboratory–support databases, schema–change requests can occur almost daily, and queries and updates that involve joins across fifteen or twenty tables would not be unusual. If Ullman's characterization of database technology is considered accurate, the HGP offers a real computational challenge to database theoreticians.

COMPUTER AND BIOLOGICAL COLLABORATIONS

The success of the HGP will depend upon advances in both biology and computer science. This duality will necessitate effective collaborations, since the there are few individuals with true knowledge in both areas. (Developing programs to train scientists for this area is yet another computational challenge.) Although such collaborations can undoubtedly be both productive and fulfilling, there are many pitfalls awaiting those who try. Avoiding the pitfalls to effect collaborations is yet another challenge. This section touches upon two problem areas.

Cultural Gaps

Differences in training create a cultural gap that can make communication between biologists and computer scientists especially difficult. To deal with diversity, biologists learn to extract signal from noise and to suppress their attention to occasional variants and problem cases. In contrast, software engineers learn to emphasize atypical cases, since these are the most likely places for software designs to fail. As a result, whenever computer scientists attempt to assess the requirements of a particular biological information system, they can be misled if biologists underestimate the complexity of their requirements by orders of magnitude. Systems analysts should recognize that when a biologist says, "All A's

are B," what may be meant is "Some A's are B, and the rest are not well understood." As a more specific example, a biologist may claim, "Of course we have to keep track of the organism from which the DNA was obtained, but that's straightforward," neglecting to mention that "keeping track of the organism" involves dealing with several million species names, synonyms, and homonyms, all of which are logically connected as a directed acyclic graph that individual researchers will most likely prefer to view as consistent spanning trees of their choosing.

The peril of being misled by biologists' inclination to simplify is real. One prominent database researcher, after meeting with an equally prominent geneticist, recently concluded that the information–handling requirements of the genome project were trivial. "He told me that representing the genome merely involved storing descriptive attributes for a set of at most 100,000 objects that must be arranged in a linear order. I told him that that could be handled by any competent undergraduate." After further discussions, that same database worker has now come to believe that unsolved database problems exist in the HGP and will likely begin directing some of his own research toward addressing these challenges.

Nomenclature Problems

Scientific nomenclature presents a special problem for database designers, since the meaning of scientific terms varies between specialties and over time. Because the databases associated with the HGP cannot be merely a snapshot of the current consensus, but rather must remain valid indefinitely, the databases must be designed to track changes in meaning. Even the most basic genetic concepts, like "gene" and "locus," mean different things to different biologists. At a recent conference, a group of bacterial geneticists were asked, "Suppose that a translocation has occurred so that all of the DNA for a given gene has been moved to a different position on the chromosome. Do we say that the gene has a new locus, or do we say that the gene's locus is at a new position?" All of the biologists claimed the question was trivial, but when pressed they split evenly in choosing an answer. If a requirements analyst had interviewed only one biologist on these definitions, the resulting system would have been perceived as inadequate by the fifty percent of biologists with differing views. Seeking the common semantic denominator introduces complexity, so that it is probably true that in order to meet the needs of many biologists, the logical atoms in a biological database should be defined at a level of detail and complexity greater than that needed by any one biologist.

Variations in concept definitions do not seem to impede the practice of biology, since biologists constantly refine their beliefs through the reality–check of experimentation. In addition, most scientists rarely read old literature and thus are unaware of the full extent of concept drift in their fields. Therefore, biologists often assert that terminological fluidity is not an issue in biological database design. This is a mistake. Many biologists don't immediately appreciate that, in a

database built with five percent error in the definition of individual concepts, a query that joins across 15 concepts has less than a 50:50 chance of returning an adequate answer. If genomic databases are going to tolerate fuzzy concepts while providing good answers to complex queries, systems much more sophisticated than textbook business databases will have to be built.

Previous thinking about formalizing a science can provide insights to the developers of scientific databases. Because every tuple in a relational database may be regarded as a formal assertion in predicate calculus about the subject domain of the database, building a genetic database bears much in common with developing a formal, axiomatic structure for genetics. Although several efforts have been made to formalize genetics, none has met with recognized success. Forty years ago, J. H. Woodger [16] made an heroic attempt to develop a formal genetic calculus, yet today no practicing geneticist is familiar with his work. Nonetheless, his observations are relevant to those building genetic databases. For example, Woodger noted that the language of geneticists is usually not as complex as their thoughts:

> Geneticists, like all good scientists, proceed in the first instance intuitively and ... their intuition has vastly outstripped the possibilities of expression in the ordinary usages of natural languages. They know what they mean, but the current linguistic apparatus makes it very difficult for them to say what they mean. This apparatus conceals the complexity of the intuitions. It is part of the business of genetical methodology first to discover what geneticists mean and then to devise the simplest method of saying what they mean. If the result proves to be more complex than one would expect from the current expositions, that is because these devices are succeeding in making apparent a real complexity in the subject matter which the natural language conceals.

This was written in 1952, before the discovery of DNA structure and the advent of molecular biology. Woodger's observations are even more applicable today.

The analysis and design efforts required to build genomic information systems will be a continuing computational challenge of the HGP. Building genomic databases without striving to ferret out, understand, decompose, and represent the underlying conceptual complexity is inviting failure. Yet most biologists consider worrying about nomenclatural details to be definitive tedium. Getting past these difficulties, to build truly useful information resources for the HGP, will tax the skills (and the patience) of computer scientists and biologists alike.

CONCLUSION

Database and computational activities are an essential part of the Human Genome Project. If this aspect is not handled well, the HGP could consume billions of dollars and researchers might still find it easier to obtain data by

repeating experiments than by querying a database. Should that happen, the project could reasonably be called a failure.

Some genomic database and software problems are fairly straightforward. Others will push the envelop of information–management theory. The HGP needs a continuum of database activities, ranging from pure application development to pure research. The research community needs production–quality, rock–solid, public–access databases right now. But research will be required to develop the new ideas and technologies necessary for the production–quality databases of a decade hence. The challenges of the Human Genome Project will drive computational science, just as earlier challenges from genetics drove the development of modern statistical analysis. (Regression analysis and analysis of variance were both initially devised, by Galton and Fisher respectively, to deal with genetic problems.)

In the Human Genome Project, computers will not merely serve as tools for cataloging existing knowledge. Rather, they will serve as instruments, helping to create new knowledge by changing the way we see the biological world. Computers will allow us to see genomes, just as radio telescopes did for quasars and electron microscopes for viruses.

REFERENCES

[1] S. Barron, M. Witten, R. Harkness, and J. Driver, "A bibliography on computational algorithms in molecular biology and genetics," *CABIOS*, vol 7, no. 2, p. 269, 1991.

[2] S. Barron, M. Witten, R. Harkness, and J. Driver, "A bibliography on computational algorithms in molecular biology and genetics," *Advances in Mathematics and Computers in Medicine*, vol. 6, in press, 1991.

[3] C. R. Cantor, "Orchestrating the human genome project," *Science*, vol. 248, pp. 49–51, 1990.

[4] M. J. Cinkosky, J. W. Fickett, P. Gilna, and C. Burks, "Electronic data publishing and GenBank," *Science*, vol. 252, pp. 1273–1277, 1991.

[5] R. R. Colwell [Ed.], *Biomolecular Data: A Resource in Transition.* New York: Oxford University Press, 1989.

[6] B. J. Culliton, "Mapping terra incognita (humani corporis)," Science, vol 250, pp. 210–212, 1990.

[7] W. Gilbert, "Towards a paradigm shift in biology," *Nature*, vol. 349, p. 99, 1991.

[8] D. K. Hawley, and W. R. McClure, "Compilation and analysis of *Escherichia coli* promoter DNA sequences," *Nucleic Acids Research*, vol. 11, pp. 2237–2255, 1983.

[9] R. Jourdain, *Programmer's Problem Solver for the IBM PC, XT, & AT*. New York: Brady Communications Company, Inc, 1986.

[10] M. L. Pearson, and D. Söll, "The human genome project: A paradigm for information management in the life sciences, *The FASEB Journal*, vol 5, pp. 35–39, 1991.

[11] J. C. Stephens, M. L. Cavanaugh, M. I. Gradie, M. L., Mador, and K. K. Kidd, "Mapping the human genome: Current status," *Science*, vol 250, pp. 237–244, 1990.

[12] United States Department of Health and Human Services, Public Health Service, National Institutes of Health, National Center for Human Genome Research, *Annual Report I—FY 1990*. Washington, DC: Government Printing Office, 1991.

[13] J. D. Ullman, *Principles of Database and Knowledge–Base Systems, Volume 1*. Rockville, Maryland: Computer Science Press, 1988.

[14] United States National Academy of Sciences, National Research Council, Commission on Life Sciences, Board on Basic Biology, Committee on Mapping and Sequencing the Human Genome, *Mapping and Sequencing the Human Genome*. Washington, DC: National Academy Press, 1988.

[15] J. D. Watson, "The human genome project: Past, Present, and Future," *Science*, vol 248, pp. 44–48, 1990.

[16] J. H. Woodger, *Biology and Language.* Cambridge: Cambridge University Press, 1952.