

■ PART I

ALGORITHMS AND MODELS

Parallel and Evolutionary Approaches to Computational Biology

NOUHAD J. RIZK

Many of the today's problems, such as those involved in weather prediction, aerodynamics, and genetic mapping, require tremendous computational resources to be solved accurately. These applications are computationally very intensive and require vast amounts of processing power and memory requirements. Therefore, to give accurate results, powerful computers are needed to reduce the run time, for example, finding genes in DNA sequences, predicting the structure and functions of new proteins, clustering proteins into families, aligning similar proteins, and generating phylogenetic trees to examine evolutionary relationships all need complex computations. To develop parallel computing programs for such kinds of computational biology problems, the role of a computer architect is important; his or her role is to design and engineer the various levels of a computer system to maximize performance and programmability within limits of technology and cost. Thus, parallel computing is an effective way to tackle problems in biology; multiple processors being used to solve the same problem. The scaling of memory with processors enables the solution of larger problems than would be otherwise possible, while modeling a solution is as much important as the computation.

In this chapter, after an introduction to genes and genomes, we describe some efficient parallel algorithms that efficiently solve applications in computational biology. An evolutionary approach to computational biology is presented based first on the search space, which is the set of all possible solutions. The second factor used for the formulation of an optimization problem is the determination of a fitness function that measures how good a particular answer is. Finally, a significant deviation from the standard parallel solution to genetic parallel algorithms approach theory is pointed out by arguing that parallel computational biology is an important sub-discipline that merits significant research attention and that combining different solution paradigms is worth implementing.

1.1 INTRODUCTION

Computational biology is the use of computational techniques to model biological systems at various levels of complexity — atomic, metabolic, cellular, and pathologic. The field of computational biology covers many areas: structural biology, biochemistry, physical chemistry, molecular biology, genomics and bioinformatics, control theory, statistics, mathematics, and computer science. Bioinformatics provides a wealth of potential challenges that can be used to advance the state of the art by creating scalable applications that can be used in customer environments. Thus, in computational biology, conducting research related to the realization of parallel/distributed scalable applications requires an understanding of the basics of all related fields. Therefore, this chapter starts with a detailed explanation of certain technical terms that have proved to be essential for researchers in computational biology.

1.1.1 Chromosome

A chromosome is a long string of double-stranded deoxyribonucleic acid (DNA), the molecule that serves as a primary repository of genetic information. Thomas Hunt Morgan found that genes on a chromosome have a remarkable statistical property, that is, genes appear as being linearly arranged along the chromosome and also that chromosomes can recombine and exchange genetic material. A gene is a unit of heredity used to describe a unit of phenotype variation.

1.1.2 Allele

Alleles are alternate forms of the same gene. There may be hundreds of alleles for a particular gene, but usually only one or a few are common. A homologous pair of chromosomes contain two alleles, one in the chromosome derived from the father and the other in the chromosome derived from the mother. If, for example, the chromosome inherited from the mother has a mutant allele at a specific position, this position on a chromosome is called a locus, and the presence of a single mutant allele creates the trait of disease. However, the child will not suffer from the disease caused by this mutation unless both the genes inherited from parents are defective or one of them is on the X chromosome, for example, hemophilia. In brief, an allele is a type of the DNA at a particular locus on a particular chromosome.

1.1.3 Recombination

Recombination or crossing over is defined as the recombination of maternal chromosome pairs with its paternal chromosome and exchanges material in the genesis of a sperm or egg. This formation of new gene combination is the result of the physical event of crossing over. The intensity of linkage of two genes can be measured by the frequency of the recombinants. The probability that a recombination event occurs between two loci is a function of the distance between these loci. In fact, the alleles at two loci that are far apart on a chromosome are more likely to combine than the

alleles that are close together on a chromosome. Genes that tend to stay together during recombination are called linked. Sometimes, one gene in a linked pair serves as a *marker* that can be used by geneticists to infer the presence of the other genes causing disease.

1.1.4 Meiosis

Before explaining meiosis, let us explain the relationship between genes and alleles. In Figure 1.1, we notice two gametes inherited from the father AD, which are called the gene 1, and the two gametes inherited from the mother ad, which are called gene 2. Therefore, the formation of haploid germ cells from diploid parent cell is called meiosis. Meiosis is informative for linkage when we identify whether the gamete is recombinant.

1.1.5 Genetic Linkage

Geneticists seek to locate genes for disorder traits (gene disease) among the genome, which is pairs of 23 human chromosomes. The statistical procedure used to trace the transmission of a disordered allele within a family is called linkage analysis. This analysis is based on genes, whose locations on a particular chromosome are already known, and are called markers [1].

Genes will be inherited together if they are close on the same chromosome because recombination is less likely. Recombinant chromosomes will occur less frequently

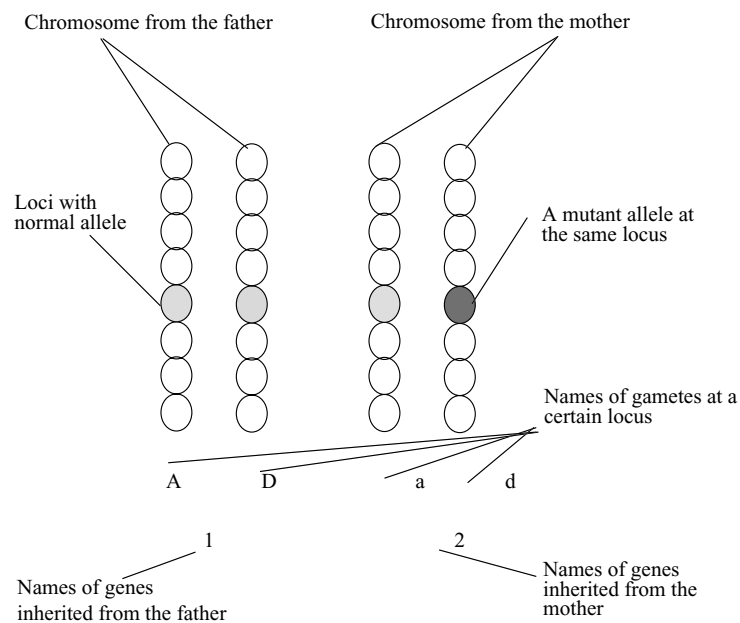


Figure 1.1 Genes and alleles at a specific locus.

TABLE 1.1 Expected Frequency of Children Through the Mother

Mother's Gametes	Recombinant	Probability
Ad	No	1
Ad	No	1
Ad	No	1
Ad	No	1

(less than half of the time) than nonrecombinant chromosomes (more than half of the time). The recombination is measured by the recombination fraction denoted by θ , which is the probability of a recombination between two loci. The lowest possible value for θ is zero, which means that there is no recombination between the marker and the trait locus. In fact, either the marker is the trait or the two loci are so close together that they rarely recombine. The upper limit of θ is 0.5, which means that the loci are not linked. The recombination frequency θ for unlinked loci is 0.5. In brief, $0 \leq \theta \leq 0.5$ [2].

The family is said to be idealized pedigree if no recombination has taken place. If there are recombinations, it is quite easy to calculate θ ; it is the summation of the number of recombinants among offspring divided by the total number of offspring. For example, if one recombinant out of nine offspring means that θ is equal to $1/9 = 0.11$.

1.1.6 Expected Frequency of Offspring

The computation of expected frequency of offspring genotypes in linkage is as follows [3]: As an example, consider a father who has haplotype AD/ad and a mother with haplotype ad/ad. All the mother's gametes will be genotyped ad. Thus, the probability that the mother gives the alleles ad is equal to 1 (see Table 1.1).

Father, however, may have one of the four different gametes, AD, Ad, aD, ad. In addition, the probability that the father gives the alleles AD is equal to the probability that the father gives allele A at marker multiplied by the probability of having no recombination between marker and trait. In fact, it is equal to $1/2(1 - \theta)$. Similarly, the probability that the father gives the alleles Ad is equal to the probability that the father gives allele A at marker multiplied by the probability of having recombination between marker and trait. In fact, it is equal to $1/2\theta$. The probability that the father

TABLE 1.2 Expected Frequency of Children Through the Father

Father's Gametes	Recombinant	Probability
AD	No	$1/2(1 - \theta)$
Ad	Yes	$1/2\theta$
AD	Yes	$1/2\theta$
Ad	No	$1/2(1 - \theta)$

TABLE 1.3 Expected Frequency of Children

Father's and Mother's Gametes	Recombinant	Probability
AD/ad	No	$1/2(1 - \theta)$
AD/ad	Yes	$1/2\theta$
aD/ad	Yes	$1/2\theta$
ad/ad	No	$1/2(1 - \theta)$

gives the alleles aD is equal to the probability that the father gives allele a at marker multiplied by the probability of having no recombination between marker and trait. In fact, it is equal to $1/2\theta$. Finally, the probability that the father gives the alleles ad is equal to the probability that the father gives allele a at marker multiplied by the probability of having recombination between marker and trait. In fact, it is equal to $1/2(1 - \theta)$ (see Tables 1.2–1.4). Then, the expected frequency among the offspring is a function of θ .

1.1.7 Multipoint Linkage Analysis

In the previous section, we assumed that we know where is the gene affected but what if we do not know? Therefore, we need to gather a large number of families in which we observe a disorder and we extract some biological specimen from each member of the family to study the linkage, but this time with many markers simultaneously; this procedure is called multipoint linkage [4]. There are two types of statistical techniques used in the linkage analysis, parametric linkage analysis and nonparametric linkage analysis. Parametric linkage analysis uses statistical procedures to estimate θ and sometimes other quantities. The odds for linkage is a quantity that is equal to the ratio of two probabilities; the numerator is the probability of observing the data given that θ is less than 0.5 (i.e., the marker and the trait loci are linked) and the denominator is the probability of observing the data given that θ is equal to 0.5 (i.e., the marker and the trait loci are not linked). The common logarithm (base 10) of the odds (likelihood) of linkage is specific to geneticists for the computation of parametric linkage analysis. It is called the LOD scores. The second method used in linkage analysis is suitable for complex gene disorders, unlike the first one suitable for single gene analysis. It is called the nonparametric approach. The advantages of nonparametric techniques are that it is not necessary to make assumptions about the mode of inheritance for the disorder; their disadvantage is they are less powerful than

TABLE 1.4 Expected Frequency Function of Theta

Offspring	Recombinant	Probability	$\theta = 0$	$\theta = 0.10$	$\theta = 0.2$	$\theta = 0.3$	$\theta = 0.4$	$\theta = 0.5$
AD	No	$1/2(1 - \theta)$	0.5	0.45	0.40	0.35	0.30	0.25
Ad	Yes	$1/2\theta$	0.00	0.05	0.10	0.15	0.20	0.25
aD	Yes	$1/2\theta$	0.00	0.05	0.10	0.15	0.20	0.25
ad	No	$1/2(1 - \theta)$	0.50	0.45	0.40	0.35	0.30	0.25

the parametric techniques. An example of nonparametric techniques is the affected sib-pair method. The geneticists gather data on a large number of sibships to locate those that have at least two members of a sibship who are affected with the disorder. The affected sib pairs are then genotyped at the marker locus, and the sib pairs are placed into one of the two mutually exclusive categories based on their genotypes at the marker. The first category includes all sib pairs who have the same genotype at the marker, these being called marker-concordant pairs. The second category is for the marker-discordant pairs, those sib pairs who have different genotypes at the marker. If the marker is not linked to the gene for the disorder, then we should expect an equal number in both categories. However, if the marker is linked to the disease locus, then there should be more marker-concordant pairs than marker-discordant pairs.

Sib Pair Analysis The sib pair analysis is the probability of having 0, 1 or 2 common alleles. This analysis is known as identity by descent (IBD) (Fig. 1.2). Consider a sib pair and suppose we wish to identify the parental origin of the DNA inherited by each sib at a particular locus, say. Label the paternal chromosomes containing the locus of interest by (a, c), and similarly label the maternal chromosomes by (b, d).

The inheritance vector of the sib-pair at the locus l is the vector $x = (x_1, x_2, x_3, x_4)$, where x

- x_1 is the label of the paternal chromosome from which sib1 inherited DNA at locus l (a),
- x_2 is the label of the maternal chromosome from which sib1 inherited DNA at locus l (b),
- x_3 is the label of the paternal chromosome from which sib2 inherited DNA at locus l (c), and
- x_4 is the label of the maternal chromosome from which sib1 inherited DNA at locus l (d).

In practice, the inheritance vector of a sibship is determined by finding enough polymorphism in the parents to be able to identify the chromosomal fragments transmitted

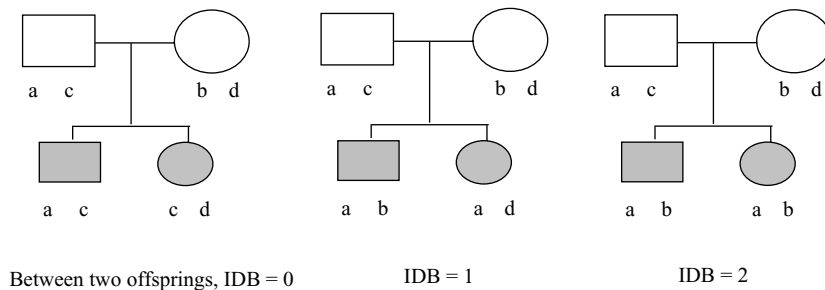


Figure 1.2 Identity by descent (IBD).

to individuals in the sibship. When IBD information is incomplete, partial information extracted from marker data may be summarized by the inheritance distribution, a conditional probability distribution over the possible inheritance vectors at the marker locus [5].

For sib-pairs, there are 16 inheritance vectors; however these are usually grouped into three distinct IBD configurations corresponding to the number of chromosomes sharing DNA IBD at the locus (Table 1.5).

Under Mendel's first law, all 16 inheritance patterns (inheritance vectors) are likely equally, hence the probabilities that two sibs share DNA IBD on 0, 1, and 2 chromosomes are $1/4$, $1/2$, and $1/4$, respectively [6].

1.1.7.1 Joint Genotype Probabilities The joint genotypes probabilities are conditioned on the number of pairs of genes (one from each individual), which are IBD. The value p_i represents the population frequency of allele i . Genes not IBD are assumed to be statically independent. Genotypes are considered to be unordered pairs of genes. For example, consider two offspring with respectively ab and cd alleles; the probability of having one common identical alleles is equal to zero, and the probability of having two common identical alleles is also equal to zero, but the probability of having zero common identical alleles is equal to the multiplication of probabilities of alleles frequency in such a population multiplied by the number of possible states $S(p_a * p_b * p_c * p_d * S)$ [1, 7].

Example of Computation: Consider $A = (1, \dots, v)$ a set of different alleles. $P_a = (p_1, \dots, p_v)$ a set of probabilities of alleles frequency. $G = (x_1, x_2, x_3, x_4)$ an ordered vector of alleles from the set A of two individual X with alleles x_1 and x_2 and the individual Y with alleles x_3 and x_4 , and s is a random gene state of G belonging to a set S of IBD states.

If $s = abcd$ implies that there is zero IBD ($m = 0$). If $s = abbc$ implies that there is one IBD (x_2 and x_3 are common). Assumed that x_1 and x_2 are never IBD and also x_3 and x_4 are never IBD. The technical definition of this assumption is that the individuals X and Y are not inbred. This implies that the set of states S contains only IBD states compatible with this assumption and the resulting states are mutually exclusive.

TABLE 1.5 Sib-Pair IBD Configurations

Number IBD	Inheritance Vectors
0	(a, b, c, d), (c, b, a, d), (a, d, c, b), (c, d, a, b)
1 (paternal)	(a, b, a, d), (c, b, c, d), (a, d, a, b), (c, d, c, b)
1 (maternal)	(a, b, c, b), (c, b, a, b), (a, d, c, d), (c, d, a, d)
2	(a, b, a, b), (a, b, a, b), (c, d, c, d), (c, d, c, d)

TABLE 1.6 Probabilities of Pairwise Genotypes Given Number of IBD

Genotypes	0 IBD pairs	1 IBD pair	2 IBD pairs
aa-aa	p_a^4	p_a^3	p_a^2
ab-aa	$2p_a^3 p_b$	$p_a^2 p_b$	0
aa-bb	$p_a^2 p_b^2$	0	0
ab-ab	$4p_a^2 p_b^2$	$p_a p_b (p_a + p_b)$	$2p_a p_b$
ab-bc	$4p_a p_b^2 p_c$	$p_a p_b p_c$	0
ab-cc	$2p_a p_b p_c^2$	0	0
ab-cd	$4p_a p_b p_c p_d$	0	0

In the first column of the Table 1.6, there are zero common alleles inherited. The relative probabilities of different states can be calculated by

$$P\{s = (x_1, x_2, x_3, x_4) \in S | m = 0\} = p_{x_1} * p_{x_2} * p_{y_1} * p_{y_2} * \text{States' cardinality.}$$

Note that x_i are all independent conditional on $m = 0$. Thus, if all the possible states for this case are only one abcd, this implies that $S = \{abcd\}$ with cardinality equal to 1. Therefore, the probability of having zero IBD between the two pairs ab-cd (row number 7) in this table is equal to $p_a * p_b * p_c * p_d * 1$.

In addition, for the two pairs ab-aa (row number 2), there are ab-aa and ba-aa, implies that $S = \{abcc, bacc\}$ with cardinality equal to 2. The probability is then $p_a * p_b * p_c * p_d * 2$.

1.1.7.2 Joint Distribution of Genotype The distribution of a set of genes is derived by enumerating all possible IBD states, then calculating the probability of each of the states [8]. These probabilities depend only on the pedigree itself, while the actual probability of observing a given genotype is calculated by conditioning on the IBD states, recognizing that non-IBD genes are conditionally independent [3]. In the example described in the previous section, the joint distribution of genotypes g given the states s belongs to $A * A * A * A * S$, where S is the cardinality of the set S :

$$P\{g = (i, j, k, j) \cap s = abcb\} = p\{x_1 = i\}p\{x_2 = j\}p\{x_3 = k\}p\{s = abcb\}.$$

$$P\{g = (i, j, k, j) \cap s = abac\} = 0.$$

As allele configuration (i, j, k, j) is incompatible with state abac. i, j, k represent alleles of unordered genotypes of two ordered individuals. This probability is noted as

$$P\{g \text{ in } (A * A * A * A) | m = k\}, \quad k = 0, 1, 2.$$

The multi-locus computation of sib-pair genotypes depends on the genotypes of parents. Parents with unknown genotypes are called founders and the offspring with known genotypes are called nonfounders.

Consider an arbitrary family (known as a pedigree) with f founders and n nonfounders. (A sib-pair pedigree is a special case with two founders and two nonfounders). The length of the inheritance vector is now $2n$ (paternal and maternal alleles of each nonfounder), and the number of possible vectors also called possible states is 2^{2n} . Denote by $a_l = (a_{l,1}, \dots, a_{l,2f})$ the vector of alleles assigned to the founders at locus. The formulation of the computation of the probability of the observed marker genotypes m_l given the inheritance vector v_l at marker locus l is

$$P\{m_l|v_l\} = \sum a_l P\{m_l, a_l|v_l\}.$$

The IBD probabilities at locus l is the summation of $P\{m_l|v_l\}$ over v_l corresponding to IBD = 0, 1 or 2.

1.1.8 Pedigree Probabilities

For a pedigree composed of k persons, each person K has an ordered genotype relative to many locations (called loci): $X_k = (x_k^1, x_k^2, \dots, x_k^l) = (x_{km}^1, x_{kp}^1, x_{km}^2, x_{kp}^2, \dots, x_{km}^l, x_{kp}^l)$ where x_{km}^i is the allele inherited from the mother at locus i and x_{kp}^i is the allele inherited from the father at locus i . The problem is to calculate the probability of phenotypes (unknown genotype) of all persons of a family, when knowing the genotype at different positions.

Another way of defining the problem, given the family with many loci in an order and the phenotypes of members of the pedigree, given also the recombination's fractions between consecutive loci θ_i , the problem will be to reconstruct the best genetic map by maximizing the chance of having data presented and to replace the old recombination fractions by the new one calculated by the algorithm. Genetic mapping is placing genes and other genetic markers on chromosomes, ordering them in relation to one another and other landmarks (such as centromeres), and assigning genetic distance to adjacent pairs.

1.1.8.1 Probability Types

We distinguish three types of probabilities:

1. *Founder Probabilities*: If the person K is a founder, the probability of the ordered genotype across all loci is equal to the multiplication of all allele frequencies at different loci. $P[X_k] = P[x_k^1] * P[x_k^2] * \dots * P[x_k^l]$.

If X_k has as alleles 1 and 3 at locus $l \rightarrow P[X_k^l = 1, 3] = P_1^l * P_3^l$. Thus, the allele frequency at locus l is the multiplication of population frequency of each allele at the specific locus.

2. *Transmission Probabilities*: This is the probability that a child inherits a particular genotype from his/her parents. $P[X_k = x_k | X_m = x_m, X_p = x_p]$, then all genotypes are ordered $P[X_k | x_m, x_p] = P[X_{km} | x_m] * P[X_{kp} | x_p]$.

3. *Penetrance Probabilities*: This is the probability of phenotypes. When genotypes are given, they are compared with phenotypes; if they are compatible then the penetrance probability is equal to 1, otherwise it is equal to zero.

1.1.8.2 Existing Methods Performing calculations on multi-locus data means calculating probabilities on pedigrees using different algorithms. The traditional algorithm is to choose new θ values by approximating the derivative of the likelihood function $L(\theta^{\text{old}})$. The main disadvantage of this method, called quasi-Newton, is that the new θ may have lower likelihood. The Expectation and Maximization (EM) algorithm overpasses this drawback by offering a powerful general approach to obtain maximum likelihood estimates from incomplete data. The algorithm starts by making an initial guess of θ distribution. Then, it calculates the expected values for complete data such as the number of recombinant and nonrecombinant meioses in each interval. Afterward, it maximizes the likelihood estimate θ^{new} . The algorithm keeps on repeating expectations and maximizations until the likelihood converges to a maximum. The theoretical advantages of EM search are: less computation time per iteration, increased likelihood on each iteration, good initial convergence properties, exact expressions for derivatives of the likelihood, and ease of generalization.

However, the problem is also stated as the construction of multi-locus linkage maps in humans or as a missing data problem. The efficient solutions are to determine the expected number of recombinations that occur in each interval, given the recombination fractions $\theta = (\theta_1, \theta_2, \dots, \theta_l)$ and the phenotypes for all loci (l_1, \dots, l_l) .

A special case reconstruction: in this case we suppose that we can observe the genotypes of each individual in a pedigree, including which alleles are on the paternally and maternally derived chromosomes. In each interval, it is then easy to find out by inspection if there is recombination or nonrecombination. Thus, the computational time is proportional to the square of the number of loci.

Elston-Stewart algorithm: this is the general case of genetic reconstruction, the algorithm is applied recursively up the family tree, with probabilities computed for each possible genotype of each child, conditional on the genotypes of his parents, the phenotype of the child, and the phenotypes for the child's descendents. The computation time scales with the number of alleles on each locus. This becomes impractical if the number of loci is more than 5.

Hidden Markov algorithm: this algorithm can be applied with any number of loci but with pedigrees of limited size. It is based on a nonhomogeneous Markov chain of inheritance vectors v_1, \dots, v_l at different loci with known transition matrices between any two consecutive loci. An inheritance vector v_i is a vector of 2^* nonfounder binary bits. A coordinate is 0 if the allele is inherited from the father; otherwise, it is 1.

Thus, the molecular biological approach to understanding the flow and expression of genetic information involves studying the structure of macromolecules (DNA, RNA, and protein) and the metabolic steps that mediate the flow of information from the genome to the phenotype of the organism. With the advent of large databases of genetic information for these macromolecules, a completely new approach to studying gene expression and its regulation has developed. This field is known as bioinformatics.

1.2 BIOINFORMATICS

Bioinformatics is a subset of a larger area of computational molecular biology which includes the application of computer and information science to areas such as genomics, mapping, sequencing, and determination of sequence and structure by classical means. The primary goals of bioinformatics are predicting three-dimensional structure from primary sequences, predicting biological or biophysical functions from either sequence or structure, and simulating metabolism and other biological processes based on these functions. Many methods of computer and information science are applied to these goals including machine learning, information theory, statistics, graph theory, algorithms, artificial intelligence, stochastic methods, simulation, logic, and so on. Bioinformatics is the recording, annotation, storage, analysis, and searching/retrieval of gene sequences, protein sequences, and structural information. Thus, the purpose of the human genome project is to map the 24 pairs of human chromosomes (including X and Y chromosomes) and delineate the location of all genes. The parallelization is dealing with gigantic amounts of data, repeated comparisons, and retrievals. But the question remains of what is the relationship between bioinformatics and computational biology.

1.2.1 Computational Biology

Computational biology is the application of computational tools and techniques to (primarily) molecular biology. It is the use of computational techniques to model biological systems at various levels of complexity — atomic, metabolic, cellular, and pathologic. It enables new ways of study in life sciences, allowing analytic and predictive methodologies that support and enhance laboratory work. It is a multidisciplinary area of study that combines biology, computer science, and statistics.

Computational biology is also called bioinformatics, although many practitioners define bioinformatics somewhat more narrowly by restricting the field to molecular biology only.

In computational biology, there are different types of problems such as how to find genes in DNA sequences, to predict the structure and functions of newly discovered proteins, to cluster proteins into families, and to align similar proteins, and to generate phylogenetic trees to examine evolutionary relationships. Therefore, there are different models of genetic change such as the long-term model that deals with the evolutionary changes among species. Another model deals with reconstructing evolutionary trees from sequences, a short-term model that deals with the genetic variations in a population and a final model that deals with finding genes by linkage and association.

1.2.2 What Drives Computational Biology?

Computational biology is one of the most rapidly growing areas of scientific computation. It is driven by two factors, the first of which is the recent explosion in the amount of available biological (particularly molecular biological) data, whereas the

second is the commercial incentive to exploit the available biological data for drug development and other discoveries.

We see, therefore, that the biological data explosion is due to the increased interest in the study of molecular biology over the past few decades, to the significant improvements in the reliability and speed of automated techniques available to gather molecular biological data over the past few years [9], and to the mapping of the human genome completed recently by both publicly and privately funded efforts (The International Human Genome Project and Celera Genomics Group). These advances have caused the amount of information available in international DNA databases to increase exponentially over the last few years. Organization, retrieval, and interpretation of this vast amount of data have become a key computational challenge.

1.2.3 Key Areas of Computational Biology

The computational biology field focuses on three key areas of computational molecular biology:

1. *Sequence Analysis*: Analyzing sequences of DNA and proteins to try to determine their structure, function, and control information. Given two sequences of characters (representing DNA or proteins), the key problem of sequence analysis is to align them to achieve the optimal match between them on a character-by-character basis. The optimality of the alignment is determined by a scoring system that gives credit for suitably matched characters and penalizes mismatches. However, gaps may be inserted between characters to try to increase the optimality of the match and therefore may lead to certain complications such as the scoring system penalizing gap insertion [10].

The advantage of aligning sequences is that, if a sequence with unknown properties aligns with a sequence with known properties, it may be possible to extrapolate the structure and function of the unknown sequence. Besides, two sequences with common ancestry may add or remove some amino acids during evolution. There are different types of sequence analysis: pairwise alignment, heuristic pairwise alignment, and multiple alignments [11–13].

2. *Structure Analysis*: Analyzing biological structures to try to determine their sequence and function, and to control information. The key problem in structure analysis is to find an optimal correspondence between the arrangements of atoms in two molecular structures (say A and B) to align them in three dimension. The optimality of the alignment is determined by using a root-mean-square measure of the distances between corresponding atoms in the two molecules. However, it is not known a priori which atom in molecule B corresponds to a given atom in molecule A (the two molecules may not even have the same number of atoms). Structure is believed to be more closely related to function of proteins than sequence. Structural alignment makes clear the common ancestry of two or more proteins (if such a common ancestry exists). It allows identification of common substructures of interest and it allows classification of proteins based on structural similarities.

3. *Function Prediction*: Understanding how sequences and structures lead to the specific functions. The key problem is to predict the function of protein structures based on sequence and structure information. The function is loosely defined, and can be thought of at many levels atomic or molecular [14, 15]. Currently, relatively little progress has been made in function prediction, particularly for higher-order processes. The current methods of function prediction are:

- *Experimentation*, used to determine the function of proteins and other structures but expensive in terms of time and money.
- *Annotation Transfer*: When sequence or structure analysis yields correspondences between structures, the known properties and function of one is used to extrapolate the properties and function of the other. This method has been extremely successful, but the facts that similar sequence or structure does not always imply similar function and that the annotated information about the “known” protein or its sequence or structure information in the database may be incomplete or incorrect are drawbacks to be considered [16].

1.2.4 On the Parallelization of Bioinformatics Applications

This section surveys the computational strategies followed to parallelize the most commonly used software in the bioinformatics arena. The studied algorithms are computationally expensive and their computational patterns range from the regular, such as database searching applications, to very irregularly structured patterns such as phylogenetic trees. Fine- and coarse-grained parallel strategies are discussed for these very diverse sets of applications. This overview outlines computational issues related to parallelism, physical machine models, parallel programming approaches, and scheduling strategies for a broad range of computer architectures.

Exploring a few general concepts about computer architectures, as well as about the parallel programming approaches that have been used to address bioinformatics applications, seems to be essential.

A parallel computer uses a set of processors that are able to cooperate in solving computational problems. This cooperation is made possible, first, by splitting the computational load of the problem (tasks or data) into parts and, second, by reconnecting the partial computations to create an accurate outcome. The way in which load distribution and reconnection (communications) are managed is heavily influenced by the system that will support the execution of a parallel application program.

Parallel computer systems are broadly classified into two main models based on Flynn’s (1972) specifications: single-instruction multiple-data (SIMD) machines and multiple-instruction multiple-data (MIMD) machines.

SIMD machines were powerful in the field of parallel computing, but now facing extinction. A typical SIMD machine consists of many simple processors (hundreds or even thousands), each with a small local memory. The complexity and often the inflexibility of SIMD machines, strongly dependent on the synchronization requirements, have restricted their use mostly to special-purpose applications.

MIMD machines are more amenable to bioinformatics. In MIMD machines, each computational process works at its own rhythm in an asynchronous fashion with complete independence of the other computational processes [17]. Memory architecture has a strong influence on the global architecture of MIMD machines, becoming a key issue for parallel execution, and frequently determines the optimal programming model.

1.2.4.1 Parallel Programming Models In simple terms, parallel software enables a massive computational task to be divided into several separate processes that execute concurrently the solution of a common task through different processors. The method used to divide tasks and rejoin the end result can be used as a point of reference to compare different alternative models for parallel programs. In particular, two key features can be used to compare models: granularity, the relative size of the units of computation that execute in parallel; and communication, the way that separate units of computation exchange data and synchronize their activity.

The finest level of software granularity is intended to run individual statements over different subsets of a whole data structure. This concept is called data-parallel, and is mainly achieved through the use of compiler directives that generate library calls to create lightweight processes called threads, and distribute loop iterations among them. A second level of granularity can be formulated as a block of instructions. At this level, the programmer identifies sections of the program that can safely be executed in parallel and inserts the directives that begin to separate tasks. When the parallel program starts, the run-time support creates a pool of threads that are unblocked by the run-time library as soon as the parallel section is reached. At the end of the parallel section, all extra processes are suspended and the original process continues to execute.

Ideally, if we have n processors, the run time should also be n times faster with respect to the wall-clock time. In real implementations, however, the performance of a parallel program is decreased by synchronization between processes, interaction (information interchanges), and load imbalance (idle processors while others are busy). Coordination between processes represents sources of overhead, in the sense that they require some time added to the pure computational workload.

Much of the effort that goes into parallel programming involves increasing efficiency. The first attempt to reduce parallelization penalties is to minimize the interactions between parallel processes. The simplest way, when it is possible, is to reduce the number of task divisions; in other words, to create coarsely-grained applications.

Once the granularity has been decided, communications are needed to enforce correct behavior and create an accurate outcome. When shared memory is available, interprocess communication is usually performed through shared variables. When several processes are working over the same logical address space, the locks, semaphores, or critical sections (blocks of code that only one process can execute at a time) are required for safe access to shared variables.

When the processors use distributed memory, sending messages over the network must perform all interprocess communication. With this message-passing paradigm, the programmer needs to keep in mind where the data is, what to communicate, and when to communicate to whom. Library subroutines are available to facilitate message-passing constructions: PVM and MPI.

1.2.4.2 Bioinformatics Applications In this section, different routinely used algorithms will be presented to describe the strategies followed to parallelize bioinformatics software. The discourse has been organized by the task-level computational pattern observed in such algorithms, from regular to irregular structure [18]. Traditionally, a regular-irregular classification, also named synchronous/asynchronous, has been used in such a way that it was closely related to the characteristic that computations were performed over dense or sparse matrices.

However, when working with nonnumerical applications, as is the case for most of the bioinformatics applications, the rate of free-dependent tasks, the data access pattern, and the task homogeneity, are appropriate indices used to classify applications.

Regular Computational Pattern: Database Searching Database searching (DBsrch) is the most heavily used bioinformatics application. It is also one of the most familiar applications to begin a discussion about parallelization in bioinformatics. It has a very simple form as far as data flow is concerned, and a broad range of strategies have been proposed to apply parallel computing.

The primary influx of information for bioinformatics applications is in the form of raw DNA and protein sequences. Therefore, one of the first steps toward obtaining information from a new biological sequence is to compare it with the set of known sequences contained in the sequence databases. Results often suggest functional, structural, or evolutionary analogies between the sequences.

DBsrch applications allow two different granularity alternatives to be considered: fine- and coarse-grained parallelism. Early approaches focused on data-parallel over SIMD machines starting with the work of Coulson et al. [8]. Deshpande et al. [20] and Jones [21] presented a work on hypercubes and CM-2 computers. Soon after, Sturrock and Collins implemented the exhaustive dynamic programming algorithm of Smith and Waterman in the MasPar family of parallel machines (from the minimum 1024 processors configuration of MP-1 systems up to a 16,384 processors MP-2 system [22, 23].)

The advantage of fine-grained approach is that this strategy only requires local communications. A coarse-grained approach is best for a great number of tasks such as most of today's parallel bioinformatics problems. However, several other applications exist for which there are not enough independent tasks to be solved concurrently. It is still possible to learn from early approaches, and obtain fruitful conclusions that improve new parallel solutions.

Enhanced algorithms may be proposed, taking into consideration the fact that when more tasks than processors are available, the simplest and most effective strategy is

coarse-grained parallelization. This is so fundamental that presenting a new algorithm with this feature goes together with its parallel coarse-grained implementation. Some good examples are as follows:

Structural Biology (Electron Microscopy), which determines viral assembly mechanisms and identifies individual proteins. The compute-intensive task in this algorithm is associated with imaging the three-dimensional structure of viruses from electron micrographs (two-dimensional projections). The number of tasks is related to the set of candidate orientations for each particle, such calculations being at different orientations, completely independent of each other.

Protein Structure Prediction: This task involves searching through a large number of possible structures representing different energy states. One of the most computationally intensive tasks calculates the solvent accessible surface area that can be measured on individual atoms if the location of neighboring atoms is known.

Searching Three-Dimensional Structure Databases: As the number of protein structures known in atomic detail increases, the demand for searching by similar structures also grows. A new generation of computer algorithms has been developed for searching by the following methods: extending dynamic programming algorithms [24]; importing strategies from computer vision areas [25] using intramolecular geometrical information, as distances, to describe protein structures [14]; and finding potential alignments based on octomeric C alpha structure fragments, and determining the best path between these fragments using a final dynamic programming step followed by least-squares superposition [26].

Linkage Analysis: Genetic linkage analysis is a statistical technique used for rapid and largely automated construction of genetic maps from gene-linkage data. One key application of linkage analysis aims to map human genes and locate disease genes. The basic computational goal in genetic linkage analysis is to compute the probability that a recombination occurs between two loci L1 and L2. Most frequently used programs estimate this recombination function by using a maximum likelihood approach [7].

All the previous examples fit perfectly to coarse-grained parallel applications, due to the large number of independent tasks and the regular computational pattern they exhibit, together with the low communication/computation rate they present. All these features make them suitable for parallelism with high efficiency rates. However, several other interesting examples have nonregular computational patterns, and they need particular strategies to better exploit parallelism.

In conclusion, it is important to integrate parallelization strategies for individual function evaluation (coarse-grained), with a strategy to parallelize the gradient estimation (fine-grained).

Semi-Regular Computational Patterns A similar problem arises in the parallelization of hierarchical multiple sequence alignments (MSA) [27–30]. The first step for solving an MSA includes calculating a cross-similarity matrix between each pair

of sequences, followed by determining the alignment topology and finally solving the alignment of sequences, or clusters themselves.

Pairwise calculation provides a natural target for parallelization because all elements of the distance matrix are independent (for a set of n sequences $n(n-1)/2$ pairwise comparisons are required [31]). Certainly, parallel strategies for the cross-matrix calculation have been proposed [32, 33], all of them in a coarse-grained approach. In addition, when the MSA is embedded in a more general clustering procedure [34], combining a dynamic planning strategy with the assignment of priorities to the different types of active tasks, using the principles of data locality, has allowed both the exploitation of the inherent parallelism of the complete applications, and the obtaining of performances that are very close to optimal. A full solution probably should combine a coarse-grained solution when computing the cross-similarity matrix with a fine-grained solution for solving the topology.

Irregular Computational Patterns Applications with irregular computational patterns are the hardest to deal with in the parallel arena. In numeric computation, irregularity is mostly related to sparse computational spaces, which introduce hard problems for data parallel distributions (fine-grained approaches) and data dependencies. The latter reduces the number of independent tasks, which affords little opportunity for developing efficient coarse-grained parallel implementations. A good example of this comes from another routine task in biological sequence analysis; that of building phylogenetic trees [35–38].

The algorithm proceeds by adding sequences into a given tree topology in such a way as to maximize the likelihood topology (suitable for coarse-grained parallelism). Once the new sequence is inserted, a local-optimization step is performed to look for minor rearrangements that could lead to a higher likelihood.

There is no generic procedure to address this type of irregular problem; hence, a good initial approach includes a detailed analysis of the computational behavior of the algorithm such as careful run-time analysis of the algorithm [34, 39].

1.2.4.3 Successful Applications of Genetic Linkage Analysis Many genetics projects were intended to develop which have at least one specific goal, which is to develop a comprehensive human genetic map, which would be used in the identification of the genes associated with genetic diseases and other biological properties and as a backbone for building physical maps. The online resources for gene mapping are very frequent on the Internet as many genome centers develop their own sets of databases to study the inherited diseases and genes and to use a multi-point analysis to locate a disease gene within a framework of markers. Early multi-point linkage analysis based on the Elston–Stewart algorithm [40] was limited to the analysis of few markers.

Lander and Green [6] described the use of hidden Markov models in calculating inheritance distribution, which allowed a more comprehensive multi-point linkage analysis and in the speed of computation subsequently implemented in the Genehunter computer program [5], which is available for download at the homepage of the Whitehead Institute for Biomedical Research [41].

In addition to rapid computation of multi-point LOD scores with dozens of polymorphic markers, the Genehunter program allows calculation of the information content for each region and the construction of multi-marker haplotypes in the pedigrees. The only limitation for the Genehunter algorithm is the size of the pedigree, which is dictated by the power of the computer. Thus, there is a need to parallelize this program. The Whitehead Institute developed a Genehunter with OpenMP support. Minor modifications were made to this program to make it thread-safe and efficient, and to be used with the parallel environment of the OpenMP.

Geneticists are always facing the limitations of the sequential Genehunter version that are memory allocation and CPU time consumption. Kuhn and Peterson from the Whitehead Institute parallelize the same sections of code from Genehunter in both Pthreads and OpenMp. Their result proves that OpenMP is a parallel programming model that facilitates parallelizing debugging and tuning bioinformatics applications, which other implementers use MPI [42, 43]. Rizk provided a new strategy for solving linkage packages analysis based on a parallel program capable of managing these two main handicaps. Conant et al. [42] parallelize the Genehunter by distributing markers among different processors — we call this algorithm the low-level parallel model. However, the high-level model is based on a master–slave paradigm. A more general strategy would be the possibility of selecting either one of these models or a combination of both. In the high-level model, the granularity of a task is one family. Future work includes the implementation of the combined model on different platforms, and using a classification of parameters so that the appropriate model of parallelism can be selected. Moreover, the linkage analysis problem can be a classification problem to predict the optimized complexity based on the selection of the appropriate model. Thus, linkage analysis problem is an application area for evolutionary algorithms.

1.3 EVOLUTIONARY COMPUTATION APPLIED TO COMPUTATIONAL BIOLOGY

Evolutionary computation is, like neural networks, an example *par excellence* for an information-processing paradigm that was originally developed and exhibited by nature and later discovered by man, who subsequently transformed the general principle into computational algorithms to be put to work on computers. Nature makes in an impressive way the use of the principle of genetic heritage and evolution. Application of the simple concept of performance-based reproduction of individuals (“survival of the fittest”) led to the rise of well-adapted organisms that can endure in a potentially adverse environment. Mutually beneficial interdependencies, cooperation, and even apparently altruistic behavior can emerge solely by evolution.

Evolutionary computation comprises the four main areas: genetic algorithms (GAs) [44], evolution strategies [45], genetic programming [46], and simulated annealing [47]. GAs and evolution strategies emerged at about the same time in the United States of America and Germany. Both techniques model the natural evolution process to optimize either a fitness function (evolution strategies) or the effort

of generating subsequent, well-adapted individuals in successive generations (GAs). Evolution strategies in their original form were basically stochastic hill-climbing algorithms and used for optimization of complex, multi-parameter objective functions that in practice cannot be treated analytically. GAs in their original form were not primarily designed for function optimization but to demonstrate the efficiency of genetic crossover in assembling successful candidates over complicated search spaces. Genetic programming takes the idea of solving an optimization problem by evolution of potential candidates one step further in that not only the parameters of a problem but also the structure of a solution is subject to evolutionary change. Simulated annealing is mathematically similar to evolution strategies. It was originally derived from a physical model of crystallization. Only two individuals compete for the highest rank according to a fitness function and the decision about accepting suboptimal candidates is controlled stochastically.

All methods presented in this chapter are heuristic, that is, they contain a random component. As a consequence (in contrast to deterministic methods) it can never be guaranteed that the algorithm will find an optimal solution or even any solution at all. Evolutionary algorithms are therefore used preferably for applications where deterministic or analytic methods fail, for example, because the underlying mathematical model is not well defined or the search space is too large for systematic, complete search (n.-p. completeness). Another application area for evolutionary algorithms that is rapidly growing is the simulation of living systems starting with single cells and proceeding to organisms, societies, or even whole economic systems [48, 49]. The goal of artificial life is not primarily to model biological life as accurately as possible but to investigate how our life or other, presumably different forms of life could have emerged from nonliving components.

Work with evolutionary algorithms bears the potential for a philosophically and epistemologically interesting recursion. At the beginning, evolution emerged spontaneously in nature. Next, man has discovered the principle of evolution and acquired knowledge on its mathematical properties. He defines GAs for computers. To complete the recursive cycle, computational GAs are applied to the very objects (DNA and proteins) from which they had been derived in the beginning.

1.3.1 Evolutionary Approaches

The explosion of biological sequence data and many of the problems posed by it require tremendous computational resources to solve exactly. Thus, many of the interesting problems arising in the analysis of biological sequence data are in the class of NP-hard. Evolutionary algorithms are one possible tool for addressing such problems. These algorithms use the techniques of survival of the fittest and natural selection to evolve solutions to particular problem instances.

Many sequence analysis problems are optimization problems. The formulation of an optimization problem is based on two factors, namely the search space which is the set of all possible solutions and the fitness function which is the measure used to determine how good a particular answer is.

The evolutionary approaches explore the search space of a problem by working from individuals. Each individual represents an encoding of a potential solution. They modify individuals using artificial operators inspired by natural mechanisms. These individuals compete on the basis of their value under the fitness function. Finally, the selected ones reproduce and live into the next generation [50].

The fitness function embodies the essential aspects of the problem to be solved. It is desirable for individuals with significant shared characteristics to have similar fitness values. The fitness function should point the GA toward the correct value, rather than away from it. In contrast, choosing a representation for a problem is a critical design decision and defines the search space. The representation specifically should help preserve the building blocks of the problem.

The interaction of each of the GA component, affecting the ability of the GA to search the space of available solutions and the design of an efficient GA to solve a particular problem, necessitates some understanding of how the individual components will work together. Their interaction is the primary driver of effective performance of the GA. The complex interactions of the GA components and the generality of the approach are both a strength and a weakness. Therefore, proper understanding of the approach allows one to avoid the weakness and exploit the strength of the GA approach.

1.3.2 Constructing an Algorithm

The work of constructing algorithms that address problems with biological relevance, that is, the work of constructing algorithms in computational biology, consists of two interacting steps. The first step is to pose a biological interesting question and to construct a model of the biological reality that makes it possible to formulate the posed question as a computational problem. The second step is to construct an algorithm that solves the formulated computational problem.

The first step requires knowledge of the biological reality, whereas the second step requires knowledge of algorithmic theory. The quality of the constructed algorithm is traditionally measured by standard algorithmic methodology in terms of the resources, most prominently time and space, it requires for solving the problem. However, as the problem solved by the algorithm originates from a question with biological relevance, its quality should also be judged by the biological relevance of the answers it produces.

The quality of an algorithm that solves a problem with biological relevance is thus a combination of its running time and space assumption and the biological relevance of the answers it produces. These two aspects of the quality of an algorithm depend on the modeling of the biological reality that led to the formulation of the computational problem that is addressed by the algorithm [51].

Constructing a good algorithm that addresses a problem with biological relevance is therefore an interdisciplinary activity that involves interchanging between modeling the biological reality and constructing the algorithm, until a reasonable balance between the running time and space assumption of the algorithm, as well as the biological relevance of the answers it produces, is achieved. The degree of interchanging between modeling and constructing of course depends on how closely related the problem addressed by the algorithm is to a specific biological application, and

therefore how relevant it is to judge the algorithm by the biological relevance of the answers it produces.

1.3.3 Application GA

There are several approaches that have been followed in field of evolutionary computation. The general term for such approaches is evolutionary algorithms. The most widely used form of evolutionary algorithm is GA.

Genetic Algorithms are unorthodox search methods. GAs were developed based on the work of John Holland and his colleagues in the 1960s and 1970s at the University of Michigan. In contrast with evolution strategies and evolutionary programming, Holland's original goal was not to design algorithms to solve specific problems, but rather to formally study the phenomenon of adaptation as it occurs in nature and to develop ways in which the mechanisms of natural adaptation might be imported into computer systems. Holland presented the GA as an abstraction of biological evolution and gave a theoretical framework for adaptation under the GAs. Genetic Algorithms have the ability to perform well over a wide range of problems. The following points make them different from other algorithms [52] in GA, variables are represented as strings of binary variables, called chromosomes, GA starts a search from a population of points instead of a single point, which avoids false peaks, GA uses the values of the objective function and GA uses probabilistic transition rules to select improved population points.

The key idea in GAs is the manipulation of chromosomes. Each chromosome represents several variables, encoded as a combination of binary digits (ones and zeroes). Hence by manipulating the chromosome, we will be able to find an optimal or near-optimal solution to the problem. Implementation of GAs requires two main functions, the creation of an initial population and the reproduction of the initial population, which leads to a new one. The reproduction itself is based on three factors: the selection of chromosomes, the crossover, and the mutation. Note that the whole reproduction process is random.

The simplest version of a GA consists of a population and a fitness function. A population of candidate represents solutions to a given problem (e.g., candidate circuit layouts), each encoded according to a chosen representation scheme (e.g., a bit string encoding a spatial ordering of circuit components). The encoded candidate solutions in the population are referred to metaphorically as chromosomes, and units of the encoding (e.g., bits) are referred to as genes. The candidate solutions are typically haploid rather than diploid.

A fitness function assigns a numerical value to each chromosome in the population, measuring its quality as a candidate solution to the problem at hand.

1.4 CONCLUSIONS

Molecular biology and computer science have grown explosively as separate disciplines. However, just as two complementary DNA strands bind together in a double

helix to better transmit genetic information, an evolving convergence has created an interrelationship between these two branches of science. In several areas, the presence of one without the other is unthinkable. Not only has traditional sequential Von Neumann-based computing been fertilized through this interchange of programs, sequences, and structures, but the biology field has also challenged high-performance computing with a broad spectrum of demanding applications (for CPU, main memory, storage capacity, and I/O response time). Strategies using parallel computers are driving new solutions that seemed unaffordable only a few years ago.

Parallel computing is an effective way to deal with some of the hardest problems in bioinformatics. The use of parallel computing schemes expands resources to the size of the problem that can be tackled, and there is already a broad gallery of parallel examples from which we can learn and import strategies, allowing the development of new approaches to challenges awaiting solution, without the need to “re-invent the wheel.”

Today, it should be natural to think in parallel when writing software, and it should be natural to exploit the implicit parallelism of most applications when more than one processor is available. In most bioinformatics applications, due to a high number of independent tasks, the simplest approaches are often the most effective. These applications scale better in parallel, are the least expensive, and are the most portable among different parallel architectures.

However, several other challenges in bioinformatics remain unsolved as far as parallel computing is concerned. They represent attractive challenges for biologists and computer scientists in the years ahead.

Evolutionary computation is an area of computer science that uses ideas from biological evolution to solve computational problems. Many such problems require searching through a huge space of possibilities for solutions, such as among a vast number of possible hardware circuit layouts for a configuration that produces desired behavior, for a set of equations that will predict the ups and downs of a financial market, or for a collection of rules that will control a robot as it navigates in its environment. Such computational problems often require a system to be adaptive, that is, to continue to perform well in a changing environment.

Thus, the genomic revolution is generating so much data in such rapid succession that it has become difficult for biologists to decipher. In particular, there are many problems in biology that are too large to be solved with standard methods. Researchers in evolutionary computation (EC) have turned their attention to these problems. They understand the power of EC to rapidly search very large and complex spaces and return reasonable solutions. While these researchers are increasingly interested in problems from the biological sciences, EC and its problem-solving capabilities are generally not yet understood or applied in the biology community.

This chapter offers a definitive resource to bridge between the computer science and biology communities. Fogel and Corne [53], well-known representatives of these fields, introduce biology and bioinformatics to computer scientists, and evolutionary computation to biologists and to computer scientists unfamiliar with these techniques.

Biological evolution is an appealing source of inspiration for addressing difficult computational problems. Evolution is, in effect, a method of searching among an

enormous number of possibilities, for example, the set of possible gene sequences, for solutions that allow organisms to survive and reproduce in their environments. Evolution can also be seen as a method for adapting to changing environments. In addition, viewed from a high level, the rules of evolution are remarkably simple: species evolve by means of random variation (via mutation, recombination, and other operators), followed by natural selection in which the fittest tend to survive and reproduce, thus propagating their genetic material to future generations.

In conclusion, problems like these require complex solutions that are usually difficult to program by hand. Artificial intelligence practitioners once believed that it would be straightforward to encode the rules that would confer intelligence on a program; expert systems were one result of this early optimism. Nowadays, however, many researchers believe that the rules underlying intelligence are too complex for scientists to encode by hand in a top-down fashion. Instead they believe that the best route to artificial intelligence and other difficult computational problems is through a bottom-up paradigm in which humans write only very simple rules and provide a means for the system to adapt. Complex behaviors such as intelligence will emerge from the parallel application and interaction of these rules. Neural networks are one example of this philosophy; evolutionary computation is another.

REFERENCES

1. J. Olson, J. Witte, and R. Elston, Tutorial in biostatistics genetic mapping of complex traits, *Statist. Med.*, 18, 2961–2981 (1999).
2. S. W. Guo, Computation of multilocus prior probability of autozygosity for complex inbred pedigrees, *Gene. Epidemiol.*, 14, 1–15 (1997).
3. R. L. Martino, C. A. Johnson, E. B. Suh, B. L. Trus, and T. K. Yap, Parallel computing in biomedical research, *Science*, 256, 902–908 (1994).
4. J. R. O’Connell and D. E. Weeks, The VITESSE algorithm for rapid exact multilocus linkage analysis via genotype set — recording and fuzzy inheritance, *Nat. Gene.*, 11, 402–408 (1995).
5. L. Kruglayk, M. J. Daly, M. P. Reeve-Daly, and E. S. Lander, Parametric and nonparametric linkage analysis: a unified multipoint approach, *Am. J. Hum. Genet.*, 58, 1347–1363 (1996).
6. E. S. Lander and P. Green, Construction of multilocus genetic linkage maps in humans, *Proc. Natl. Acad. Sci. USA*, 84, 2363–2367 (1987).
7. J. Ott, *Analysis of Human Genetic Linkage*, The Johns Hopkins University Press, Baltimore (revised edition), 1991.
8. A. B. Smith Cedric and A. Stephens David, Simple Likelihood and Probability Calculations for Linkage Analysis, in I. H. Pawlowitzki, J. H. Edwards, and E. A. Thompson, Eds., *Genetic Mapping of Disease Genes*, Academic Press, London, 1997, pp. 73–96.
9. J. Hodgson, Gene sequencing’s industrial revolution, *IEEE Spectrum*, 38, 37–42 (2000).
10. S. B. Needleman and C. D. Wunsch, A general method applicable to the search for similarities in the amino acid sequence of two proteins, *J. Mol. Biol.*, 48, 443–453 (1970).

11. S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, Basic local alignment search tool, *J. Mol. Biol.*, 215, 403–410 (1990).
12. S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman, Gapped BLAST and PSI-BLAST: a new generation of protein DB search programs, *Nucl. Acids Res.*, 25 (17), 3389–3402 (1997).
13. A. Krogh, M. Brown, I. S. Mian, K. Sjölander, and D. Haussler, Hidden Markov models in computational biology: applications to protein modeling, *J. Mol. Biol.*, 235, 1501–1531 (1994).
14. L. Holm and Ch. Sander, Searching protein structure databases has come of age, *Proteins*, 19, 165–173 (1994).
15. D. J. Lipman and W. R. Pearson, Rapid and sensitive protein similarity searches, *Science*, 227, 1435–1441 (1985).
16. P. Bork, T. Dandekar, Y. Diaz-Lazcoz, F. Eisenhaber, M. Huynen, and Y. Yuan, Predicting function: from genes to genomes and back, *J. Mol. Biol.*, 283, 707–725 (1998).
17. Hwang Kai and Xu Zhiwei, *Scalable Parallel Computing: Technology, Architecture, Programming*, McGraw-Hill Series in Computer Engineering, 1998.
18. A. Rodriguez, L. G. de la Fraga, E. L. Zapata, J. M. Carazo, and O. Trelles, Biological Sequence Analysis on Distributed-Shared Memory Multiprocessors, in *Proceedings of the 6th Euromicro Workshop on Parallel and Distributed Processing*, Madrid, Spain, 1998.
19. A. F. W. Coulson, J. F. Collins, and A. Lyall, Protein and nucleic acid sequence database searching: a suitable case for parallel processing, *Comput. J.*, 39, 420–424 (1987).
20. A. S. Deshpande, D. S. Richards, and W. R. Pearson, A platform for biological sequence comparison on parallel computers, *CABIOS*, 7, 237–247 (1991).
21. R. Jones, Sequence pattern matching on a massively parallel computer, *CABIOS*, 8, 377–383 (1992).
22. T. F. Smith and M. S. Waterman, Identification of common molecular subsequences, *J. Mol. Biol.*, 147, 195–197 (1981).
23. S. S. Sturrock and J. Collins, MPsrch version 1.3, *BioComputing Research Unit*, University of Edinburgh, UK, 1993.
24. C. A. Orengo, N. P. Brown, and W. T. Taylor, Fast structure alignment for protein databank searching, *Proteins*, 14, 139–167 (1992).
25. D. Fisher, O. Bachar, R. Nussinov, and H. Wolfson, An efficient automated computer vision based technique for detection of three-dimensional structural motifs in proteins, *J. Biomol. Struct. Dyn.*, 9, 769–789 (1992).
26. I. N. Shindyalov and P. E. Bourne, Protein structure alignment by incremental combinatorial extension (CE) of the optimal path, *Protein Eng.*, 11 (9), 739–747 (1998).
27. F. Corpet, Multiple sequence alignments with hierarchical clustering, *Nucl. Acid Res.*, 16, 10881–10890 (1988).
28. O. Gotoh, Optimal alignment between groups of sequences and its application to multiple sequence alignment, *CABIOS*, 9 (2), 361–370 (1993).
29. W. Miller, Building multiple alignments from pairwise alignments, *CABIOS*, 9 (2), 169–176 (1993).
30. J. D. Thompson, D. G. Higgins, and T. J. Gibson, Clustal W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting position-specific gap penalties and weight matrix choice, *Nucl. Acids Res.*, 22, 4673–4680 (1994).

31. O. Trelles, E. L. Zapata, and J. M. Carazo, Mapping Strategies for Sequential Sequence Comparison Algorithms on LAN-Based Message Passing Architectures, in *Lecture Notes in Computer Science, High Performance Computing and Networking*, Springer-Verlag, Berlin, Vol. 796, 1994, pp. 197–202.
32. G. H. Gonnet, M. A. Cohen, and S. A. Benner, Exhaustive matching of the entire protein sequence database, *Science*, 256, 1443–1445 (1992).
33. S. Date, R. Kulkarni, B. Kulkarni, U. Kulkarni-Kale, and A. Kolaskar, Multiple alignment of sequences on parallel computers, *CABIOS*, 9 (4), 397–402 (1993).
34. O. Trelles, M. A. Andrade, A. Valencia, E. L. Zapata, and J. M. Carazo, Computational space reduction and parallelization of a new clustering approach for large groups of sequences, *Bioinformatics*, 14 (5), 439–451 (1998).
35. M. Gribskov and J. Devereux, *Sequence Analysis Primer*, UWBC Biotechnical Resource Series, 1991.
36. L. L. Cavalli-Sforza and A. W. F. Edwards, Phylogenetic analysis: models and estimation procedures, *Am. J. Hum. Genet.*, 19, 233–257 (1967).
37. J. Felsenstein, Maximum-likelihood estimation of evolutionary trees from continuous characters, *Soc. Hum. Genet.*, 25, 471–492 (1973).
38. J. Felsenstein, Phylogenies from molecular sequences: inference and reliability, *Annu. Rev. Genet.*, 22, 521–565 (1988).
39. C. Ceron, J. Dopazo, E. L. Zapata, J. M. Carazo, and O. Trelles, Parallel implementation for DNAmI program on message-passing architectures, *Parallel Comput. Appl.*, 24 (5–6), 701–716 (1998).
40. R. C. Elston and J. Stewart, A general model for the genetic analysis of pedigree data, *Hum. Heredity*, 21, 523–542 (1971).
41. Whitehead Institute for Biomedical Research. <http://www-genome.wi.mit.edu>.
42. G. C. Conant, S. J. Plimpton, W. Old, A. Wagner, P. R. Fain, and G. Heffelfinger, Parallel genehunter: implementation of a linkage analysis package for distributed-memory architectures, *J. Parallel Distrib. Comput.*, 63, 674–682 (2003).
43. N. Rizk, Parallelization of IBD computation for determining genetic disease maps, *Concurrency Comput. Pract. Exp.*, 16, 933–943 (2004).
44. J. Holland, Genetic algorithms and the optimal allocations of trials, *SIAM J. Comput.*, 2 (2), 88–105 (1973).
45. I. Rechenberg, Bioinik, evolution und optimierung, *Naturwissenschaftliche Rundschau*, 26, 465–472 (1973).
46. J. Koza, *Genetic Programming*, MIT Press, 1993.
47. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, Optimization by simulated annealing, *Science*, 220 (4598), 671–680 (1983).
48. S. M. Le Grand and K. M. Merz, The application of the genetic algorithm to the minimization of potential energy functions, *J. Global Optimization*, 3, 49–66 (1993).
49. S. Y. Lu and K. S. Fu, A sentence-to-sentence clustering procedure for pattern analysis, *IEEE Trans. Sys., Man Cybernetics*, 8, 381–389 (1978).
50. D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, 1989.
51. H.-P. Schwefel, *Numerical Optimization of Computer Models*, John Wiley, Chichester, (originally published in 1977), 1981.

52. S. Schulze-Kremer and A. Levin, Search for Protein Conformations with a Parallel Genetic Algorithm: Evaluation of a Vector Fitness Function, Final Report and Technical Documentation of Protein Folding Application (Work package 1), European ESPRIT project # 6857 PAPAGENA, 1994.
53. G. Fogel and D. Corne, *Evolutionary Computation in Bioinformatics*, Morgan Kaufmann, 2002.
54. A. Whittmore and J. Halpern, Probability of gene identity by descent: computation and applications, *Biometrics*, 50(1), 109–117 (1994). Department of Health Research and Policy, Stanford University School of Medicine, California, 94305.