INFORMS Journal on Computing
Vol. 16, No. 3, Summer 2004, pp. 211–231
ISSN 0899-1499 | EISSN 1526-5528 | 04 | 1603 | 0211

DOI 10.1287/ijoc.1040.0073
© 2004 INFORMS

Opportunities for Combinatorial Optimization in Computational Biology

Harvey J. Greenberg
Mathematics Department, University of Colorado at Denver, P.O. Box 173364, Denver, Colorado 80217-3364, USA, harvey.greenberg@cudenver.edu

William E. Hart
Discrete Algorithms and Mathematics, Sandia National Laboratories, Albuquerque, New Mexico 87185-1110, USA, wehart@sandia.gov

Giuseppe Lancia
Departimento di Matematica e Informatica, Università di Udine, Via delle Scienze 206, 33100 Udine, Italy, lancia@dimi.uniud.it

This is a survey designed for mathematical programming people who do not know molecular biology and want to learn the kinds of combinatorial optimization problems that arise. After a brief introduction to the biology, we present optimization models pertaining to sequencing, evolutionary explanations, structure prediction, and recognition. Additional biology is given in the context of the problems, including some motivation for disease diagnosis and drug discovery. Open problems are cited with an extensive bibliography, and we offer a guide to getting started in this exciting frontier.

Key words: computational biology; combinatorial optimization; global optimization; integer programming; minimum energy; bioinformatics; molecular structure prediction; protein folding; protein alignment; rearrangements; assembly; sequence alignment; SNP; sorting by reversals

History: Accepted by Edward A. Wasil, area editor for Feature Articles; received August 2002; revised March 2003, September 2003; accepted January 2004.

1. Introduction

Although computational biology has been an increasing activity in computer science for more than a decade, it has been only the past few years that optimization models have been developed and analyzed by researchers whose primary background is operations research (OR). The purpose of this survey is to demonstrate the applicability of mathematical programming, from an OR perspective, to these problems in molecular biology.

We begin with some vocabulary, but more, context-dependent biology will be described in each section. This will not be enough biology to develop your own research in this exciting frontier, but it is enough to understand many of the problems and make an informed decision whether to pursue this field. The appendix offers guidance to getting started.

One class of problems involves sequences from one of the following alphabets:

1. Four nucleotides in DNA (one-letter code in bold capital):

{Adenine, Cytosine, Guanine, Thymine}

2. Four nucleotides in RNA (one-letter code in bold capital):

{Adenine, Cytosine, Guanine, Uracil}

3. Twenty amino acid residues in proteins:

Name	Symbol	Name	Symbol
Alanine	A	Leucine	L
Arginine	R	Lysine	K
Asparagine	N	Methionine	M
Aspartic acid	D	Phenylalanine	F
Cysteine	C	Proline	P
Glutamine	Q	Serine	S
Glutamic acid	E	Threonine	T
Glycine	G	Tryptophan	W
Histidine	H	Tryosine	Y
Isoleucine	I	Valine	V

From one sequence's information, we would like to recognize or predict structure. From multiple sequences, we would like to compare structures and determine if they are in the same "family." It is believed, with some reservation, that structure determines function, although this issue is still being explored. If we can predict function from sequence, we

can design and test drugs *in silico* (in the computer), rather than *in vivo* (in an organism or living cell), or *in vitro* (in a test tube).

The process by which this works is the *central dogma* of biology:

$$\text{DNA} \overset{\text{transcription}}{\Longrightarrow} \text{RNA} \overset{\text{translation}}{\Longrightarrow} \text{proteins.}$$

There is evidence that these processes usually obey the *thermodynamics hypothesis*, which states that matter seeks a minimum free-energy state. That is one basis for using optimization to predict or analyze structures, but it is not the only use. For example, using statistical methods one needs to maximize a likelihood function or minimize an error function, each subject to certain conditions.

In general, proteins with similar structures have similar functions, although this is not always true. More recently, attention has been paid to interactions of proteins, under the general topic of pathway inference (Bower and Bolouri 2001, Greenberg et al. 2002). A logical extension is more general interactions, modeled with gene networks and protein complexes. One approach to this is systems biology (Ideker et al. 2001), taking into account some of the many biological complexities.

In the following sections we give mathematical programming models for a broad spectrum of applications. We begin with *sequence alignment*, which has been well surveyed (Błażewicz et al. 1997). More recent work in single nucleotide polymorphisms (SNPs) and haploids uses linear programming and combinatorial optimization models. Then, we consider two problems in rearranging or assembling DNA segments. We also consider the biology problem that is perhaps the most celebrated: protein folding. Here we describe two fundamental applications in protein science: structure prediction and structure comparison.

We present problem formulations for each of these applications and discuss how solutions can be obtained. The problems are mostly NP-hard, so exact algorithms are limited in the size of problem they can solve. This raises challenging opportunities for finding approximation algorithms. Some metaheuristics have been applied, but there are opportunities to experiment further.

Standard mathematical programming terminology is used, which can be found in the *Mathematical Programming Glossary* (Greenberg 2003).

2. Sequence Alignment

Sequence alignment is the association of each member of one sequence with a member of one or more other sequences in a way that maximizes some measure of similarity. The sequence of three characters ABC

ABC-- -BCDE	A-BC BCDE	AB-C BCDE	ABC- BCDE
(a)	(b)	(c)	(d)

Figure 1 **Example Alignments of Two Sequences,** ABC **and** BCDE

can be aligned with the sequence BCDE by aligning the BC in common and having spaces inserted into the sequences for the complete alignment. That is the first alignment in the example alignments shown in Figure 1.

The next section presents the concepts that give the exact meaning of the spaces, how we measure the similarity, and why we want to do this.

2.1. Concepts

One reason to do this is to infer properties of one sequence, given that we know those properties of the other. For example, we might know where the genes are in one DNA sequence, but not in the other. If the second sequence is sufficiently similar, we can infer where its genes are. This approach is productive because individual gene sequences in mouse and human are highly similar; even the linear order of the genes is similar in the two genomes. The mouse genome can be viewed as a set of under 200 segments of the human genome whose order has been permuted.

More generally, there are segments of interest that we are trying to understand by using information from the first sequence. An implication of this is that we can discover binding sites that would enable us to alter cellular processes—such as a drug for changing a gene that produces a malfunctioning protein, or that produces too much or too little of a protein.

The sequences could be from any of our four alphabets, but we shall focus on the DNA alphabet, $\mathscr{A} = \{\text{A,C,G,T}\}$. We let $\mathscr{A}^+$ denote the set of non-null sequences from $\mathscr{A}$.

Given two sequences (s, t) of the same length, drawn from the same alphabet, their *Hamming distance*, $D_H(s, t)$, is the number of positions in which they have different characters.

Example:				
	sequence s:	AAT	AGCAA	AGCACACA
	sequence t:	TAA	ACATA	ACACACTA
	$D_H(s,t) =$	2	3	6

A *similarity measure* is a function that associates a numerical value with a pair of sequences. Often this is normalized to be in $[0, 1]$. When comparing two sequences, we introduce operations whose meaning stems from evolution and choose those operations that seek to maximize the similarity of the sequences. The notion of similarity is dual to that of distance: greater similarity corresponds to less distance between two sequences.

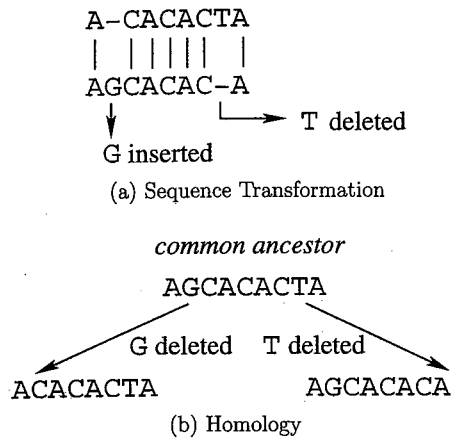

Figure 2 **Insertion and Deletion to Align Sequences**

Similarity measures vary, but they are generally more flexible than distance measures. For one thing, similarity does not require two sequences to have the same length. Also, we allow operators such as shifting positions. For the last example above, s and t differ by six characters. However, if we are allowed to delete G from s and T from t, the two sequences become ACACACA. In this sense, they are only two characters apart.

We consider a sequence alignment problem that uses three operations:

1. Delete a character from one sequence.
2. Insert a space into one sequence.
3. Replace one character with another.

A biological meaning of deletion and insertion is illustrated in Figure 2. The first assumption, shown in (a), is that one sequence descended from the other. Another view is by homology: their similarity is based on having a common ancestor in evolution, shown in (b).

In case (a), we do not want to assume which sequence was earlier, so we tacitly assume time symmetry. This implies that insertion into one sequence is a deletion in the other, depending upon which evolved into the other. Computationally, we view it as a shift in one sequence or the other. In the example all other characters matched, so there were no replacements.

2.2. Pairwise Alignment

The problem of pairwise alignment is to find operations on s and t, where the symbol "-" is added to the alphabet to represent deletion or insertion, such that the Hamming distance is minimized. For example, the alignment in Figure 1(a) has a Hamming distance of three, and others each have a Hamming distance of four, so (a) would be the preferred alignment. In general, let s^i denote the subsequence $(s_1, \ldots, s_i)$, with $s^0 \stackrel{\text{def}}{=} \phi$ (=null sequence). Define t^j similarly. Let $c(a, b)$ denote the cost of replacing a with b at some point in the sequence, where $a \neq b$; let $c(-, b)$ denote the cost of an insertion and let $c(a, -)$ denote the cost of a deletion. The *unit cost model* is where $c(a, -) = c(-, b) = 1$, independent of the characters a and b.

Let $D(s^i, t^j)$ denote the minimum total cost of applying these operations; this is sometimes called the *edit distance* between sequences s^i and t^j. In particular, the unit cost model is simply the Hamming distance, which is the number of evolutionary events that occurred to transform one sequence into the other, or to transform a common ancestor into each of them. Further, let L_s denote the length of string s. Then, D satisfies the following recursion:

$$D(s^i, t^j) = \begin{cases} D(s^{i-1}, t^{j-1}) & \text{if } s_i = t_j \\ \min \left\{ \begin{array}{ll} c(s_i, t_j) + D(s^{i-1}, t^{j-1}) & \text{(replacement)} \\ c(-, t_j) + D(s^{i-1}, t^j) & \text{(insertion)} \\ c(s_i, -) + D(s^i, t^{j-1}) & \text{(deletion)} \end{array} \right\} & \text{if } s^i \neq \phi \text{ and } t^j \neq \phi \\ L_{t^j} \quad \text{if } s^i = \phi; \; L_{s^i} \text{ if } t^j = \phi. \end{cases}$$

Beginning with $D(\phi, \phi) = 0$, this dynamic program (DP) can be solved in $O(L_s L_t)$ time.

A variety of cost functions have been considered by researchers. It is generally held that four consecutive insertions are preferred to four non-consecutive insertions. The rationale for this is that a consecutive sequence of insertions (or deletions) could result from a single evolutionary event. A consecutive sequence of insertions is called a *gap* with length L equal to the number of insertions after the first one. The cost to begin a gap is α, and the cost of the gap is βL, where $\alpha > \beta$. The *affine gap* model seeks to minimize $\sum_i (\alpha + \beta L_i)$, where the sum is over the gaps.

Example:	ACGTCCACG	ACGTCCACG
	A-G-CCACG	A---CCACG
Cost:	2α	$\alpha + 2\beta$

Gusfield's system, XPARAL (Gusfield 2001), provides a complete parametric analysis of α and β.

2.3. Multiple Sequence Alignment (MSA)

Dynamic programming provides a fast algorithm to align two sequences for a variety of distance measures. However, several approaches and alignment measures have been proposed to align $k > 2$ sequences. Aligning multiple sequences consists in arranging them as a matrix having each sequence in a row. This is obtained by possibly inserting spaces in each sequence so that the expanded sequences all have the same length. One measure of distance is to

sum the edit distances of the pairwise alignments. For example consider the sequences

Example: s: AAGCTGCAAAGC
t: AAGCTG-AAAGC
u: A-GCT-CAA-GC
v: A-GCTTCAACCG

The total edit distance of these sequences is $D_H(s,t)+D_H(s,u)+D_H(s,v)+D_H(t,u)+D_H(t,v)+D_H(u,v)$. Using the unit cost model, this becomes

$$1+3+5+4+6+4=23.$$

(Note: $u_2=v_2=$ - is considered a match.)

The complexity of a DP algorithm to minimize the total edit distance is $O(2^k L_1,\ldots,L_k)$. A small problem has $k=10$ and $L_i=100$ for $i=1,\ldots,k$. The DP for the associated problem has complexity 782×10^{20}, which is greater than the postulated age of the universe in microseconds! Thus, DP is not practical. Further, MSA is NP-hard for this objective (Wang and Jiang 1994), so we should not expect any exact algorithm to solve this problem efficiently. Insights into some of the alignment heuristics can be found by studying Clustal (Thompson et al. 1995).

Another approach to MSA considers the sequences as leaves of a phylogenetic tree—a graphical representation of the evolutionary history of species or their parts. Solving MSA involves finding sequences for the internal nodes so as to minimize the sum of the pairwise distances associated with edges. The simplest of these is the Steiner tree, an example of which is shown in Figure 3. Only one node (s_0) is to be determined, and the sequence is to minimize the sum, $\sum_{i=1}^{4} D(s_0,s_i)$.

Jiang et al. (1994) used Steiner tree alignment in an approximation algorithm for an arbitrary tree, which guarantees being within twice the minimum total distance. (See Gusfield 1997 for more on string structures and algorithms.)

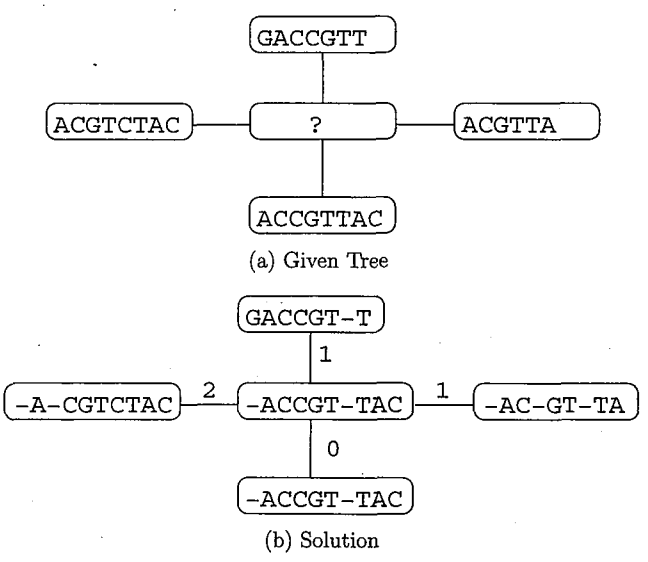

Figure 3 **Steiner Tree Alignment Example: Total Distance = 4**

2.4. Open Questions

A primary method for obtaining approximate solutions to MSA is the Carrillo-Lipman (Carrillo and Lipman 1988, Fuellen 1997) use of projection, using dynamic programming in two-dimensional state spaces to infer bounds. Gusfield (1993) gives a simple proof of an approximation algorithm, which guarantees being no worse than twice (in fact $2-2/k$) the sum of pairwise alignments (for any distance function). (What we call an objective function is typically called a *scoring function*, whose value is the *score*, of a candidate alignment.) One open question is whether there exists a better approximation algorithm, i.e., a δ-approximation algorithm with $\delta<2$. Bafna et al. (1997) present an approximation algorithm with ratio $2-l/k$, where l is any constant smaller than k. Although this is better than Gusfield's $2-2/k$ ratio, as k grows, the approximation ratio still tends to two.

Notredame and Higgins (1996) present a genetic algorithm for MSA, but little has been done using metaheuristics since then. This is therefore another open question: *Can metaheuristics obtain reliably good solutions to large MSA problems*? By "reliably" and "good," we mean with the kind of results we experience in well-established OR problems, like scheduling.

3. SNPs and Haploids

3.1. Concepts

The process of passing from the sequence of nucleotides in a DNA molecule to a string over the DNA alphabet is called *sequencing*. While in principle this may seem a simple step, merely preliminary to all the computational problems discussed here, the truth is that sequencing is a crucial phase, and has not yet been fully resolved. A sequencer is a machine that is fed some DNA and whose output is a string of As, Ts, Cs, and Gs. To each letter, the sequencer attaches a value (confidence level) that essentially represents the probability that the letter has been correctly read (the "base has been correctly called").

The main problem with sequencing is that current technology is not able to sequence a long DNA molecule, which must therefore first be replicated (cloned) into many copies and then be broken at random, into several pieces (called *fragments*) of about 1,000 nucleotides each, which are individually fed to a sequencer. The cloning phase is necessary so that the fragments can have nonempty overlap. From the overlap of two fragments one may infer a longer fragment, and so on, until the original DNA sequence

has been reconstructed. This is, in essence, the principle of *shotgun sequencing*, in which the fragments are assembled back into the original sequence by using sophisticated algorithms and powerful computers. Shotgun sequencing allowed an early completion of the sequencing of the human genome (Venter et al. 2001, Genome 2001). The assembly (i.e., overlap and merge) phase is complicated by the fact that in a genome there exist many regions with identical content (called *repeats*) scattered all around and due to replicating events during evolution. The repeats may fool the assembler into thinking that they are all copies of the same region. The situation is complicated further from the fact that diploid genomes are organized into pairs of chromosomes (a paternal and a maternal copy), which may have identical or nearly identical content, a situation that makes the assembly process even harder.

To partly overcome these difficulties, the fragments used in shotgun sequencing may have some extra information attached. In fact, they are obtained by a process that generates pairs (called *mate pairs*) of fragments instead of individual ones, with a fairly precise estimate of the distance between them. These pairs are guaranteed to come from the same copy of a chromosome (either both from the maternal or both from the paternal copy) and may help whenever one of them comes from a repeat region while the other does not (and can be used as an anchor to place its troublesome mate).

The recent whole-genome sequencing efforts have confirmed that the genetic makeup of humans (as well as other species) is remarkably well conserved, and we all share some 99% identity at the DNA level. Hence, small regions of differences must be responsible for our diversities. The smallest possible region, consisting of a single nucleotide, is called *single nucleotide polymorphism*, or SNP (pronounced "snip"). It is believed that SNPs are the predominant form of human genetic variation, so their importance cannot be overestimated for medical, drug-design, diagnostic, and forensic applications.

Broadly speaking, a polymorphism is a trait, common to everybody, whose value can be different but drawn in a limited range of possibilities, called *alleles* (for a simple example, think of the blood group). A SNP is a particular nucleotide site, placed in the middle of a DNA region that is otherwise identical in everybody, at which we observe a statistically significant variability. In particular, a SNP is a polymorphism of only two alleles (out of the four possible), for which the less frequent allele is found in the population with some nonnegligible frequency, usually taken to be 5%. Because DNA of diploid organisms is organized in pairs of chromosomes, for each SNP one can either be *homozygous* (same allele on both chromosomes) or *heterozygous* (different alleles). The values of a set of SNPs on a particular chromosome copy define a *haplotype*. The haplotyping problem is to determine a pair of haplotypes, one for each copy of a given chromosome, that provides full information of the SNP fingerprint for an individual at that chromosome. In Figure 4 we give a simple example of a chromosome with three SNP sites. The individual is heterozygous at SNPs 1 and 3 and homozygous at SNP 2. The haplotypes are CCA and GCT.

```
Chrom. c, paternal: ataggtccCtatttccaggcgcCgtatacttcgacgggActata
Chrom. c, maternal: ataggtccGtatttccaggcgcCgtatacttcgacgggTctata

Haplotype 1 →          C          C          A
Haplotype 2 →          G          C          T
```

Figure 4 A Chromosome and the Two Haplotypes

In recent years, several optimization problems have been defined for SNP data. In the remainder of this section, we address the haplotyping problem for a single individual and for a set of individuals (a population). In the first case, the input is inconsistent haplotype data. Note that, for diploid organisms, the two copies of each chromosome are sequenced together, are not identical, and, as previously observed, there are unavoidable sequencing errors. In the latter case, the input is ambiguous genotype data, which specifies only the multiplicity of each allele for each individual (i.e., it is known if individual i is homozygous or heterozygous at SNP j, for each i and j).

3.2. Haplotyping a Single Individual

As discussed in the introductory remarks, sequencing produces either individual fragments or pairs of fragments (mate pairs) that both come from one of the two copies of a chromosome. Even with the best possible technology, sequencing errors are unavoidable: these consist of bases that have been miscalled or skipped altogether. Further, contaminants can be present, i.e., DNA coming from another organism that was wrongly mixed with the one that had to be sequenced. In this framework, the *haplotyping problem for an individual* can be informally stated as follows:

> Given inconsistent haplotype data coming from fragment sequencing, find and correct the errors from the data to retrieve a consistent pair of SNPs haplotypes.

Depending on what type of errors one is after, there can be many versions of this problem. In Lancia et al. (2001a), the minimum fragment removal (MFR) and minimum SNP removal (MSR) problems are considered, which we briefly discuss here.

Given the fact that at each SNP only two alleles are possible, we can encode them by using a binary alphabet. In the sequel, the two values that a SNP can take are denoted by the letters A and B. A haplotype,

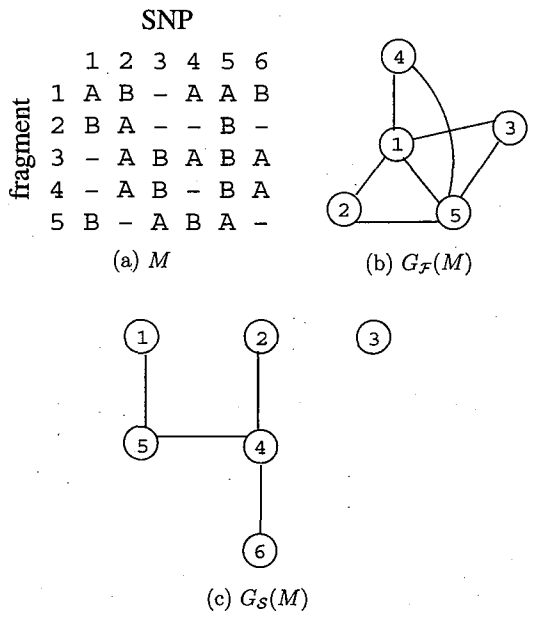

Figure 5 **Conflict Graphs for Haplotype Matrix, *M***

i.e., a chromosome content projected on a set of SNPs, is then simply a string over the alphabet {A, B}.

The basic framework for a single-individual haplotyping problem is as follows. There is a set $\mathcal{S} = \{1, \ldots, n\}$ of SNPs and a set $\mathcal{F} = \{1, \ldots, m\}$ of fragments. Each SNP is covered by some of the fragments, and can take the values A or B. Hence, a SNP i is defined by a pair of disjoint subsets of fragments, A_i and B_i. Because there is a natural ordering of the SNPs, given by their physical location on the chromosome, the data can be represented by an $m \times n$ matrix over the alphabet {A, B, -}, called the *SNP matrix*, defined in the obvious way. The symbol "-" is used to represent a SNP not covered by a fragment.

A gapless fragment is one covering a set of consecutive SNPs (i.e., the As and Bs appear consecutively in that row). We say that a fragment has k gaps if it covers $k+1$ blocks of consecutive SNPs. There can be gaps for two reasons: (i) thresholding of low-quality reads (if the sequencer cannot call a SNP A or B with enough confidence); (ii) mate-pairing in shotgun sequencing. Particularly important is the case $k = 1$, which is equivalent to two gapless fragments coming from the same chromosome. This is the case of mate pairs, used for shotgun sequencing.

Two fragments i and j are said to be *in conflict* if there exists a SNP k such that $(i \in A_k, j \in B_k)$ or $(i \in B_k, j \in A_k)$. This means that either i and j are not from the same chromosome copy, or there are errors in the data. Given a SNP matrix M, the fragment conflict graph is the graph $G_{\mathcal{F}}(M) = (\mathcal{F}, E_{\mathcal{F}})$ with an edge for each pair of fragments in conflict. Two SNPs, i and j, are said to be in conflict if A_i, B_i, A_j, and B_j are all nonempty and there exist two fragments u and v such that the submatrix defined by rows u and v and columns i and j has three symbols of one type (A or B) and one of the opposite (B or A respectively). It is easy to see that two SNPs in conflict imply that there are three fragments forming an odd cycle in the graph $G_{\mathcal{F}}(M)$. Hence the two SNPs cannot both be correct (assuming the fragments are). Given a SNP matrix M, the SNP conflict graph is the graph $G_{\mathcal{S}}(M) = (\mathcal{S}, E_{\mathcal{S}})$, with an edge for each pair of SNPs in conflict.

If $G_{\mathcal{F}}(M)$ is a bipartite graph, $\mathcal{F}$ can be segregated into two sets H_1 and H_2 of pairwise compatible fragments. From each set one can infer one haplotype by fragment overlap (this process is known as *phasing*). Note that the overlap may not be unique because there may be a subset of fragments that do not overlap with any of the remaining fragments. We call a SNP matrix M *feasible* if $G_{\mathcal{F}}(M)$ is bipartite. Note that a SNP matrix for error-free data must be feasible. Hence, the optimization problems to be defined correct a SNP matrix so that it becomes feasible.

The following optimization problems arose in the context of sequencing the human genome and are studied in Lancia et al. (2001a), Lippert et al. (2002):

- MFR: Given a SNP matrix, remove the minimum number of fragments (rows) so that the resulting matrix is feasible.
- MSR: Given a SNP matrix, remove the minimum number of SNPs (columns) so that the resulting matrix is feasible.

The first problem is mainly suited for a situation in which, more than sequencing errors, one is worried about the presence of contaminants. The second problem is more suited in the presence of sequencing errors only, when all the fragments are to be retained. These problems were shown (Lancia et al. 2001a) to be polynomial for gapless data (M is a gapless matrix if each row is a gapless SNP). The main connection between them is given by the following theorem.

THEOREM 3.1. *Let M be a gapless SNP matrix. Then, $G_{\mathcal{F}}(M)$ is a bipartite graph if, and only if, $G_{\mathcal{S}}(M)$ is a stable set.*

A *stable set*, also called an *independent set*, is a subset of nodes in a graph such that no two nodes are adjacent (see Greenberg 2003 for details and related terms). Because all SNP conflicts must be eliminated, a feasible solution to the MSR problem needs to remove nodes from $G_{\mathcal{S}}(M)$ until a stable set remains. The optimal solution must leave the largest stable set. When M is gapless, the problem is solvable in polynomial time, as was shown in Lancia et al. (2001a) by proving that in this case $G_{\mathcal{S}}(M)$ is a perfect graph. (It is known that finding the largest stable set in any perfect graph is a polynomial problem Golumbic 1980, Groetschel et al. 1984.) A simpler proof than the one in Lancia et al. (2001a) is as follows. Let $Q = (\mathcal{S}, A)$

with $\{i, j\} \in A$ if i is not in conflict with j and $i < j$ (as column indices in M). Because it can be shown that, for any three SNPs $u < v < w$, if u is not in conflict with v and v is not in conflict with w then also u is not in conflict with w, Q is a comparability graph, and hence perfect (Golumbic 1980). However, $G_{\mathcal{S}'}(M)$ is the complement of Q, so it is also perfect.

A later theoretical improvement in Rizzi et al. (2002) extended these results to fragments with gaps of bounded length, giving $O(2^{2l}m^2n + 2^{3l}n^3)$ dynamic programming algorithms for MFR and $O(mn^{2l+2})$ for MSR for instances with gaps of total length l. This algorithm is hardly practical, however, on instances for which the gaps can be rather large, such as in the presence of mate pairs. The problems were shown to be NP-hard in general (Lancia et al. 2001a). Because gaps generally occur, there is a need for practical algorithms for the gapped versions of these problems.

The following version of the haplotyping problem appears to be the most appropriate to account for the fact that data come from sequencing, and hence there are sequencing (read) errors to be corrected.

> MLF (*minimum letter flips*): Given a SNP matrix, flip the minimum number of letters (A into B and vice versa) so that the resulting matrix is feasible.

This problem has not been studied so far, and hence it is open as far as its complexity and availability of practical algorithms. Particularly interesting is its weighted version in which, for each entry, there is a weight for its flipping, which should be proportional to (or at least correlated with) the confidence level of the sequencer machine's reading errors. Another variant contemplates a three-way flip (i.e., one can flip from/to the gap as well).

Consider a square in M (i.e., four symbols at the corners of two rows and two columns) with no gaps. Call such a square *even* if there is an even number of As (and hence also Bs) and *odd* otherwise. If a solution does not use flips to eliminate a whole column (by making it all A or all B), in any even (odd) square an even (odd) number of letters must be flipped.

Hence, the following version of the set covering problem, called the *parity set covering problem*, may be useful to solve the MLF problem: Given a family $F = E \cup O$ of elements partitioned into even elements E and odd elements O and a collection $\mathcal{C}$ of subsets of F, find a set $\mathcal{C}' \subseteq \mathcal{C}$ such that each $e \in E$ belongs to an even number of elements of $\mathcal{C}'$ (possibly none), each $o \in O$ belongs to an odd number of elements of $\mathcal{C}'$, and $|\mathcal{C}'|$ is minimum. To our knowledge, this version of the set covering problem has not been studied.

3.3. Haplotyping a Population

Haplotype data are particularly sought after in the study of complex diseases (those affected by more than one gene), because they contain complete information about which set of gene alleles are inherited together. However, because polymorphism screens are conducted on large populations, in such studies, it is not feasible to examine the two copies of each chromosome separately, and genotype, rather than haplotype, data are usually obtained. A genotype describes the multiplicity of each SNP allele for the chromosome of interest. At each SNP, three possibilities arise: Either one is homozygous for the allele A, or homozygous for the allele B, or heterozygous (a situation that we shall denote with the symbol X). Hence a genotype is a string over the alphabet {A, B, X}, where each position of the letter X is called an *ambiguous* position. For a genotype g and SNP j, let $g[j]$ denote the j th symbol of g. We say that a genotype g is *resolved* by the pair of haplotypes $\{h, q\}$ if, for each SNP j, $g[j] = \mathtt{A}$ implies $h[j] = q[j] = \mathtt{A}$, $g[j] = \mathtt{B}$ implies $h[j] = q[j] = \mathtt{B}$, and $g[j] = \mathtt{X}$ implies $h[j] \neq q[j]$. We then write $g = h \oplus q$. A genotype is called *ambiguous* if it has at least two ambiguous positions (a genotype with at most one ambiguous positions can be resolved uniquely). A genotype g is said to be *compatible* with a haplotype h if h agrees with g at all unambiguous positions. The following inference rule, given a genotype g and a compatible haplotype h, defines q such that $g = h \oplus q$.

Inference Rule. Given a genotype g and a compatible haplotype h, obtain a new haplotype q by setting $q[j] \neq h[j]$ at all ambiguous positions and $q[j] = h[j]$ at the remaining positions.

The *haplotyping problem for a population* is the following.

> Given a set $\mathcal{G}$ of m genotypes over n SNPs, find a set $\mathcal{H}$ of haplotypes such that each genotype is resolved by at least one pair of haplotypes in $\mathcal{H}$.

To turn this problem into an optimization problem, one has to specify the objective function. We describe here two possible objective functions. On the study of the first formulation, although most natural, there has been little progress thus far. In this formulation, one seeks to minimize $|\mathcal{H}|$ (see Gusfield 2003 for an integer programming approach to this problem). Thinking of a haplotype h as a set, which covers all genotypes that are compatible with it, this is similar to the set covering problem, with the additional constraint that each ambiguous genotype g must be covered by (at least) two sets h and q for which $g = h \oplus q$.

The second formulation of the haplotyping problem has been studied (Gusfield 1999, 2000) and is based on a greedy algorithm for haplotype inference, also known as *Clark's rule*.

In its second formulation, the problem began as a feasibility problem. We ask if the haplotypes in $\mathcal{H}$ can be obtained by successive applications of the inference rule, starting from the set of haplotypes

obtained by resolving the unambiguous genotypes (of which it is assumed here there is always at least one). This way of proceeding was proposed in Clark (1990) with arguments from theoretical population genetics in support of its validity. In essence, Clark's rule is the following, nondeterministic, algorithm. Let $\mathcal{G}'$ be the set of non-ambiguous genotypes, and let $\mathcal{H}$ be the set of haplotypes obtained unambiguously from $\mathcal{G}'$. Start with setting $\mathcal{G} \leftarrow \mathcal{G}\backslash\mathcal{G}'$ (notation: $\mathcal{G}\backslash\mathcal{G}'$ is the set of elements in $\mathcal{G}$ minus those in $\mathcal{G}'$, and this replaces $\mathcal{G}$). Then, repeat the following. Take a $g \in \mathcal{G}$ and a compatible $h \in \mathcal{H}$ and apply the inference rule, obtaining q. Set $\mathcal{G} \leftarrow \mathcal{G}\backslash\{g\}$, $\mathcal{H} \leftarrow \mathcal{H} \cup \{q\}$, and iterate. When no such g and h exist, the algorithm will have succeeded if $\mathcal{G} = \varnothing$ and will have failed otherwise.

For example, suppose $\mathcal{G} = \{\mathtt{XAAA}, \mathtt{XXAA}, \mathtt{BBXX}\}$. The algorithm starts by setting $\mathcal{H} = \{\mathtt{AAAA}, \mathtt{BAAA}\}$ and $\mathcal{G} = \{\mathtt{XXAA}, \mathtt{BBXX}\}$. The inference rule can be used to resolve XXAA from AAAA, obtaining BBAA, which can, in turn, be used to resolve BBXX, obtaining BBBB. However, one could have started by using BAAA to resolve XXAA obtaining ABAA. At that point, there would be no way to resolve BBXX. The non-determinism in the choice of the pair g, h to which we apply the inference rule can be settled by fixing a deterministic rule based on the initial sorting of the data. Clark (1990) used a large (but tiny with respect to the total number of possibilities) set of random initial sortings to run the greedy algorithm on real and simulated data sets, and reported the best solution overall. Many times the algorithm failed, but Clark's algorithm can be viewed as a heuristic for the optimization version of the problem: Find the ordering of application of the inference rule that leaves the fewest number of unresolved genotypes in the end. This problem was studied by Gusfield (1999), who proved it is NP-hard and APX-hard. (A problem is APX-hard if there is a constant $\alpha > 1$ such that the existence of an α-approximation algorithm would imply P = NP. See Ausiello et al. (1999) for a full description of the class APX.)

As for practical algorithms, Gusfield (2000) proposed an integer programming approach for a graph-theoretic formulation of the problem. The problem is first transformed (by an exponential-time reduction) into a problem on a digraph $G = (N, A)$, defined as follows. Let $N = \bigcup_{g\in\mathcal{G}} N(g)$, where $N(g) := \{(h, g)\colon h \text{ is compatible with } g\}$. $N(g)$ is (isomorphic to) the set of possible haplotypes obtainable by setting each ambiguous position of a genotype to one of the two possible values. Let $N' = \bigcup_{g\in\mathcal{G}'} N(g)$ be (isomorphic to) the subset of haplotypes unambigously determined from the set $\mathcal{G}'$ of unambiguous genotypes. For each pair $v = (h, g')$, $w = (q, g)$ in N, there is an arc $(v, w) \in A$ if g is ambiguous, $g' \neq g$, and $g = h \oplus q$ (i.e., q can be inferred from g via h). Then, any directed tree rooted at a node $v \in N'$ specifies a feasible history of successive applications of the inference rule starting at node $v \in N'$. The problem can then be stated as: find the largest number of nodes in $N\backslash N'$ that can be reached by a set of node-disjoint directed trees, where each tree is rooted at a node in N' and where for every ambiguous genotype g, *at most* one node in $N(g)$ is reached.

The above graph problem was shown to be NP-hard (Gusfield 1999). (Note that the reduction of the haplotyping problem to this one is exponential time, and hence it does not imply NP-hardness trivially.) For its solution, Gusfield proposed

$$\max \sum_{v\in N\backslash N'} x_v\colon x_v \in \{0,1\} \quad \forall v \in N$$

$$\sum_{v\in N(g)} x_v \le 1 \quad \forall g \in \mathcal{G}\backslash\mathcal{G}'$$

$$x_w \le \sum_{v\colon (v,w)\in A} x_v \quad \forall w \in N\backslash N'.$$

Gusfield focused on the LP relaxation because he observed, on real and simulated data, that its solution was almost always integer. He reduced the LP dimension by defining variables for only those nodes that are reachable from nodes in N'. In cases where the LP relaxation was not integer, Gusfield manually set one or more fractional variables equal to zero. He also pointed out the possibility of an integer solution containing directed cycles, but this situation never occurred in his experiments.

A simple improvement of this model is immediate for researchers with mathematical programming background. First, one could add subtour elimination-type inequalities, which are easy to separate. Second, the model could be solved within a standard branch-and-cut framework, of which Gusfield's current approach explores only the root node.

3.4. Using SNPs for Disease Diagnosis

We finish this section by pointing out a problem that arises in the context of disease diagnosis (for instance, when trying to determine a gene mutation responsible for a tumor).

Genotype data are collected from a population of m individuals, of which some have a certain disease while the others do not. Hence the data comprise a set $\mathcal{G}$ of m genotypes over n SNPs. Let $\mathcal{G} = \mathcal{G}_D \cup \mathcal{G}_H$ where $\mathcal{G}_D$ is the subset of genotypes from people with the disease and $\mathcal{G}_H$ are the genotypes from healthy people. Assuming the disease is due to "faulty" haplotypes, the following new version of the haplotyping problem, called the *disease haplotyping problem*, is to be considered. Resolve the ambiguous genotypes into a set $\mathcal{H}$ of haplotypes for which there exists a subset $\mathcal{H}_D \subseteq \mathcal{H}$ such that: (i) for all $g \in \mathcal{G}_D$, there exist $h, q \in \mathcal{H}$ such that $g = h \oplus q$ and $h \in \mathcal{H}_D \vee q \in \mathcal{H}_D$; (ii) for all

$g \in \mathcal{G}_H$, there exist $h, q \in \mathcal{H}$ such that $g = h \oplus q$ and $h \notin \mathcal{H}_D \wedge q \notin \mathcal{H}_D$. The objective calls for minimizing $|\mathcal{H}|$. When there is no feasible solution, the objective function should be changed into the following: Find $\mathcal{H}$ and $\mathcal{H}_D \subseteq \mathcal{H}$ for which the total number of genotypes not satisfying (i) or (ii) above is minimum.

We also mention the *minimum informative subset* of SNPs, which is defined as a subset S of SNPs, of minimum possible size, such that, projecting the data over the SNPs in S, each genotype in $\mathcal{G}_D$ is different from each genotype in $\mathcal{G}_H$. In this case, it is possible that a diagnostic test for the disease could be limited to checking the SNPs in S. In a slightly different version of this problem, all the projected genotypes are required to be different from each other. If there are no ambiguous genotypes in the data, then this version is easily shown to be the NP-hard minimum test set problem (problem [SP96] in Garey and Johnson 1979) that is solvable by a reduction to the set covering problem.

The *haplotype tagging* problem is to pick a subset of SNP sites such that the state of those SNP sites specifies the full haplotype. This has received a lot of attention from biologists lately, and picking a smallest such set of tag sites can be solved by set covering methods. This computation is practical for the sizes of current data sets.

4. Genome Rearrangements

4.1. Introduction

With the large amount of genomic data that have become available in the past decade, it is now possible to try and compare the genomes of different species to find their differences and similarities. This is a very important problem because, when developing new drugs, we typically test them on mice before humans. But how close is a mouse to a human? How much evolution separates the two species?

Although there are very effective algorithms for comparing two DNA sequences, no such general algorithm exists for comparing two genomes. (One exception is MUMmer Delcher et al. 1999, 2002; which is based on sequence alignment, using an advanced implementation of suffix trees Kurtz 1999.) In principle, one could consider a genome as a very long string (3 billion letters in humans) and use the sequence alignment algorithm, but there are two good reasons for not doing this. First, the time required would be very large, even for the low-degree polynomial alignment algorithm. Second, and more importantly, the model of sequence alignment is inappropriate for genome comparisons, where differences should be measured not in terms of insertions/deletions/mutations of single nucleotides, but rather rearrangements of long DNA regions, which occurred during evolution.

Consider the following example. Using sequence comparison, it is almost impossible to find a similarity between the two sequences

$$s_1 = \text{GGAATGGTTTCACTTCCC}$$

$$s_2 = \text{GGCCCTTCACTTTGGTAA}.$$

However, a single event explains how the second sequence is related to the first: s_2 can be obtained by reversing s_1, except for the first two letters. Reversing part of a sequence is one of the many evolutionary events possible, by which long chunks of DNA are moved around in a genome. These events happen mainly in the production of sperm and egg cells (but also for environmental reasons), and have the effect of rearranging the genetic material of parents in their offspring. When such mutations are not lethal, after a few generations they can become stable in a population. In this case, we talk of *speciation*, meaning that a new species was derived from another.

The main evolutionary events known are deletions, duplications, transpositions, inversions, and translocations. These events affect a long fragment of DNA on a chromosome. In a deletion the fragment is simply removed from the chromosome. A duplication creates many copies of the fragment, and inserts them in different positions, on the same chromosome. When an inversion or a transposition occurs, the fragment is detached from its original position and then is reinserted, on the same chromosome. In an inversion, it is reinserted at the same place, but with opposite orientation than it originally had. In a transposition, it keeps the original orientation but ends up in a different position. Finally, a translocation causes a pair of fragments to be exchanged between the ends of two chromosomes. Figure 6 illustrates these events, where each string represents a chromosome.

Because evolutionary events affect long DNA regions (several thousand bases), the basic unit for comparison is not the nucleotide, but rather the gene. In fact, the computational study of rearrangement problems started after it was observed that several species share the same genes (i.e., the genes have identical, or nearly identical, DNA sequences), however differently arranged. For example, most genes of the mitochondrial genome of *Brassica oleracea* (cabbage) are identical in *Brassica campestris* (turnip), but appear in a completely different order. Much of the pioneering work in genome rearrangement problems is due to Sankoff and his colleagues (beginning with Sankoff et al. 1990).

The general genome comparison problem is as follows:

> Given two genomes (i.e., two sets of sequences of genes) find a sequence of evolutionary events that, applied to the first genome, turn it into the second.

```
CATTttataggttagCTTGTTAATCTC
                ↓                    (Deletion)
        CATTCTTGTTAATCTC

    TGTTAcgttcTTGTTAAGGTTAG
                ↓                    (Duplication)
TGTTAcgttcTTGTcgttcTAAGGcgttcTTAG

ATTCTTgttttataGGCTAGATCCGCCATGGA
                ↓                    (Transposition)
ATTCTTGGCTAGATCCGCgttttataCATGGA

   ATTCTTGTTttataggttagAATTTG
                ↓                    (Inversion)
   ATTCTTGTTgattggatattAATTTG

CTGTGGATgcaggacat    TCATTGAaataa
                ↓                    (Translocation)
CTGTGGATaataa    TCATTGAgcaggacat
```

Figure 6 Five Types of Evolutionary Events

Under a general parsimony principle, the solution sought is the one requiring the minimum possible number of events. A weighted model, based on the probability of each event, would be more appropriate, but these probabilities are very hard to determine. It is an open problem in genome comparison to develop sound mathematical models and algorithms for the general version of the problem. In fact, in the past decade, people have concentrated on evolution by means of some specific event alone, and have shown that these special cases can be already very hard to solve (Bafna and Pevzner 1996, 1998; Caprara 1999b; Caprara et al. 2001; Kececioglu and Ravi 1995; Kececioglu and Sankoff 1995).

The two events that have received more attention are inversions and transpositions, so we will focus on them for the remainder of this section. Inversions are considered the predominant of all types of rearrangements. For historical reasons, they have become known as *reversals* in the computer science community. Because reversals and transpositions are single-chromosome rearrangements, what we call genome, from here on, has to be understood as a particular chromosome of a given genome. Two genomes are compared by looking at their common genes. After numbering each of n common genes with a unique label in $\{1, \ldots, n\}$, each genome is a permutation of the elements $\{1, \ldots, n\}$. Let $\pi = (\pi_1, \ldots, \pi_n)$ and $\sigma = (\sigma_1, \ldots, \sigma_n)$ be two genomes. By possibly relabeling the genes, we can always assume that $\sigma = \alpha := (12, \ldots, n)$, the identity permutation. Hence, the problem becomes turning π into α.

A *reversal* is a permutation ρ_{ij}, with $1 \le i < j \le n$, defined as

$$\rho_{ij} = \Big(1 \cdots i-1 \underbrace{\boxed{j\, j-1 \cdots i+1\, i}}_{\text{reversed}} j+1 \cdots n\Big).$$

Note that by applying (multiplying) ρ_{ij} to a permutation π, one obtains

$$(\pi_1 \cdots \pi_{i-1},\ \pi_j\ \pi_{j-1} \cdots \pi_i,\ \pi_{j+1} \cdots \pi_n),$$

i.e., the order of the elements $\pi_i, \ldots, \pi_j$ has been reversed. Let $\mathcal{R} = \{\rho_{ij}\colon 1 \le i < j \le n\}$. $\mathcal{R}$ is a set of *generators* of S_n, the set of the $n!$ permutations of $\{1, \ldots, n\}$. That is, each permutation π can be expressed (nonuniquely) as a product $\alpha\rho^1\rho^2 \cdots \rho^D$ with $\rho^i \in \mathcal{R}$ for $i = 1, \ldots, D$. The minimum value D such that $\alpha\rho^1\rho^2 \cdots \rho^D = \pi$ is called the *reversal distance* of π, and denoted by $d_R(\pi)$. *Sorting by reversals* (SBR) is the problem of finding $d_R(\pi)$ and a sequence $\rho^1\rho^2, \ldots, \rho^{d_R(\pi)}$ that satisfies $\alpha\rho^1\rho^2 \cdots \rho^D = \pi$.

The above formulation does not consider the fact that a reversal not only changes the order of some genes, but it causes the nucleotide sequence of each reversed gene to be complemented. To account for this situation, a genome can be represented by a signed permutation, i.e., a permutation in which each element is signed either "+" or "−." For a signed permutation, the effect of a reversal is not only to flip the order of some consecutive elements, but also to complement their sign. For instance, the reversal ρ_{24} applied to the signed permutation $(+1\ -4\ +3\ -5\ +2)$ yields the signed permutation $(+1\ +5\ -3\ +4\ +2)$. *Signed sorting by reversals* (SSBR) determines the minimum number of reversals that turn a signed permutation π into $(+1\ +2\ \cdots\ +n)$. Note that, for signed permutations, reversals of a single element are allowed.

The problem of signs does not arise in transpositions, where the direction of each gene is always preserved. A transposition is a permutation defined by i, j, k, with $1 \le k < i \le j \le n$, as $\tau_{ijk} = (1 \cdots k-1,\ i\ i+1 \cdots j,\ k \cdots i-1, j+1 \cdots n)$. Applying τ_{ijk} to π has the effect of moving the strip of elements $\pi_i \cdots \pi_j$ from their position to the position immediately before π_k. Let $\mathcal{T}$ be the set of all possible transpositions. $\mathcal{T}$ is a set of generators of S_n so that each permutation π can be expressed (nonuniquely) as a product $\alpha\tau^1\tau^2 \cdots \tau^D$ with $\tau^i \in \mathcal{T}$ for $i = 1, \ldots, D$. The minimum value D such that $\alpha\rho^1\rho^2 \cdots \rho^D = \pi$ is called the *transposition distance* of π, and denoted by $d_T(\pi)$. *Sorting by transpositions* (SBT) is the problem of finding $d_T(\pi)$ and a sequence $\tau^1\tau^2, \ldots, \tau^{d_T(\pi)}$ that satisfies $\alpha\rho^1\rho^2 \cdots \rho^D = \pi$.

4.2. Sorting by Reversals

The study of sorting by reversals began with its unsigned version. The first exact branch-and-bound method, suitable for only small problems ($n \leq 30$), is due to Kececioglu and Sankoff (1995). A major step towards the practical solution of the problem was made by Bafna and Pevzner (1996), who, building on the previous results by Kececioglu and Sankoff, found a nice combinatorial characterization of π in terms of its breakpoints. A *breakpoint* is given by a pair of adjacent elements in π that are not adjacent in α—that is, there is a breakpoint at position i, if $|\pi_i - \pi_{i-1}| > 1$.

The analysis of breakpoints provides the key to bounding $d_R(\pi)$ effectively. Let $b(\pi)$ denote the number of breakpoints. Then, a trivial bound is $d_R(\pi) \geq \lceil b(\pi)/2 \rceil$ because a reversal can remove at most two breakpoints, and α has no breakpoints. However, Bafna and Pevzner showed how to obtain a significantly better bound from the breakpoint graph $G(\pi)$, which has a node for each element of π and edges of two colors, say red and blue. Red edges connect elements π_i and π_{i-1} for each position i at which there is a breakpoint, and blue edges connect h and k whenever $|h-k| = 1$, but h and k are not adjacent in π. $G(\pi)$ can be decomposed into a set of edge-disjoint color-alternating cycles. Let $c(\pi)$ be the maximum number of edge-disjoint alternating cycles in $G(\pi)$. Bafna and Pevzner proved the following theorem: For every permutation π, $d_R(\pi) \geq b(\pi) - c(\pi)$.

The lower bound $b(\pi) - c(\pi)$ turns out to be very tight, as observed first experimentally by various authors and then proved to be almost always the case by Caprara (1999a), who showed that determining $c(\pi)$ is essentially the same problem as determining $d_R(\pi)$ (Caprara 1999b). Moreover, Caprara proved both problems to be NP-hard, thus settling a long-standing open question about the complexity of unsigned SBR. SBR was later shown by Berman and Karpinski (1999) to be APX-hard as well. The best approximation algorithm known is by Berman et al. (2002) and achieves a ratio of 1.375.

The NP-hardness of computing $c(\pi)$ may seem like a major drawback against the use of the lower bound $b(\pi) - c(\pi)$ for the practical solution of SBR. However, this is not the case. In fact there is an effective integer linear programming (ILP) formulation to find $c(\pi)$ and for any upper bound $c'(\pi)$ to $c(\pi)$, also the value $b(\pi) - c'(\pi)$ is a lower bound to $d_R(\pi)$. Based on the aforementioned ILP formulation, a good upper bound to $c(\pi)$ is obtained by LP relaxation. The following is an ILP formulation to find $c(\pi)$.

Let $\mathscr{C}$ denote the set of all the alternating cycles of $G(\pi) = (V, E)$, and for each $C \in \mathscr{C}$ define a binary variable x_C. The following is the ILP formulation of the maximum cycle decomposition:

$$\max\left\{\sum_{C\in\mathscr{C}} x_C : \sum_{C\ni e} x_C \leq 1, \forall e \in E, x_C \in \{0,1\}, \forall C \in \mathscr{C}\right\}. \quad (1)$$

A good upper bound to $c(\pi)$ can be obtained by LP relaxation. Based on these ideas, Caprara et al. (2001) presented a branch-and-price algorithm whose latest version can routinely solve, in a matter of seconds, instances with $n = 200$ elements, a large enough size for all real-life instances available so far. Note that no effective ILP formulation has ever been found for modeling SBR directly. Finding such a formulation constitutes an interesting theoretical problem.

The LP relaxation of (1) has an exponential number of variables, but it can be solved in polynomial time by column-generation techniques. Solutions of some non-bipartite, perfect matching problems are used to price the variables.

Today, SBR is regarded as practically solved, being one of the few NP-hard problems for which a (worst-case) exponential algorithm (namely, branch-and-price) is fairly good on most instances. The situation is even better as far as the optimization of SSBR is concerned. In fact, with a deep combinatorial analysis of the cycle decomposition problem for the permutation graph of a signed π, SSBR was shown to be polynomial by Hannenhalli and Pevzner (1995). This result came as a surprise, at a time when unsigned SBR was still an open problem, and the two versions of the problem were expected to have the same complexity. The original $O(n^4)$ algorithm of Hannenhalli and Pevzner for SSBR was improved over the years to an $O(n^2)$ algorithm for finding the optimal solution (Kaplan et al. 1997) and an $O(n)$ algorithm (Bader et al. 2001) for finding the signed reversal distance (but not a sequence of reversals that achieve this distance).

The other evolutionary events in Figure 6 do not have such a rich theory. The analogous problems for sorting by transpositions or translocations are particularly interesting, and there are no results comparable to the algorithmic framework built on breakpoint analysis. Thus, this provides another opportunity to advance this field. Once we can deal with each of the evolutionary events individually, we must then consider combinations.

While sorting by reversals is well developed, there is an open problem of another kind, which concerns reversals by a stochastic process. That is the subject of the next section.

4.3. Expected Reversal Distance

Because the optimization of the reversal distance has become a well-understood problem, research has

shifted to the study of the expected reversal distance, a problem for which there has been no similar success. In a probabilistic model of evolution, one can assume that each reversal is a random event, and evolution follows a random walk in a graph having a node for each possible genome. Define the reversal graph as the graph $G_R = (S_n, E)$ in which there is an edge between each pair of permutations π, σ such that σ can be obtained from π by a reversal. SBR corresponds to the shortest-path problem in G_R. If π and ρ represent two genomes, $d_R(\pi\rho^{-1})$ gives a lower bound to the number of evolutionary events that occurred in the evolution of π and ρ from a common ancestor.

However, the actual path followed by evolution does not necessarily correspond to the shortest path, and a more reliable scenario can be based on a probabilistic analysis. Assume each reversal can occur with the same probability for a permutation. Starting at the identity permutation (denoted α), consider a random walk in G_R in which, at each step, an edge is chosen with uniform probability among all edges incident on the current node. Let Y_k be a random variable representing the reversal distance of the permutation obtained after k steps of the random walk. Perhaps the most significant open problem for sorting by reversals, both signed and unsigned, is to determine the expected value $E[Y_k]$, or tight lower and upper bounds.

The *Cayley graph* (Babai 1991) of a group $\mathcal{G}$ with respect to a set $\mathcal{T}$ of generators is a graph with vertex set $\mathcal{G}$ and edges $\{g, gr\}$ ($g \in \mathcal{G}, r \in \mathcal{T}$). Because the set of all reversals is a set of generators of S_n, the graph G_R is a (non-bipartite) Cayley graph. For this graph, the probability of ending at node v after k steps of a random walk approaches the random distribution, as k increases. In Caprara and Lancia (2000), it is shown that $O((n \log n)^{O(1)})$ random reversals are enough to end at a practically random permutation. Because Bafna and Pevzner (1996) show that for a random permutation π, $E[d_R(\pi)] \geq (1 - 4/\log n)n$, the same bound holds for $E[Y_k]$, with k large enough. However, genomic permutations should not be random, and hence it is important to derive a similar bound for $E[Y_k]$ for small values of k. A first step in this direction was achieved by considering another random variable, X_k, the number of breakpoints in an unsigned permutation after k uniform random reversals, that is correlated with Y_k. Caprara and Lancia (2000) proved that $E[X_k] = (n-1)(1 - ((n-3)/(n-1))^k)$ and have shown how this value can be used to derive a better (i.e., closer to the true sequence of events) solution than the optimal SBR solution, when $d_R \leq n/2$. These results were generalized to a whole class of genomic distances, for signed and unsigned permutations and circular genomes as well, in Wang and Warnow (2001).

5. Protein Structure Prediction and Recognition

Proteins are complex biological macromolecules that are composed of a sequence of amino acids, which is encoded by a gene in a genome. Proteins are key elements of many cellular functions. Fibrous proteins contribute to hair, skin, bone, and other fibrous parts. Membrane proteins stay in a cell's membrane, where they mediate the exchange of molecules and information across cellular boundaries. Water-soluble globular proteins serve as enzymes that mediate and catalyze most of the biochemical reactions that occur in cells.

There are 20 different amino acids specified in the genetic code. Amino acids are joined end-to-end during protein synthesis by the formation of peptide bonds (see Figure 7). The sequence of peptide bonds forms a "main chain" or "backbone" for the protein, off of which project the various side chains.

The functional properties of proteins depend upon their three-dimensional structures. Understanding and predicting these structures has proven quite daunting, despite the fact that the structures of thousands of proteins have been determined (Berman et al. 2000). Unlike the structure of other biological macromolecules (e.g., DNA), proteins have complex, irregular structures. Our focus in this section is on globular proteins, which exhibit a specific native state.

The sequence of residues in a protein is called its *primary structure*. A variety of structural motifs have been identified for proteins. Proteins exhibit a variety of secondary structure motifs that reflect common

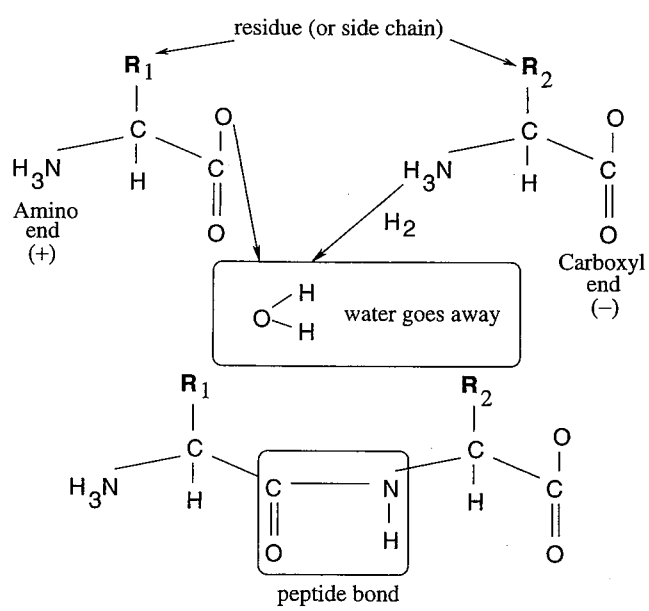

Figure 7 **The Peptide Bond Joining Two Amino Acids When Synthesizing a Protein**

structural elements in a local region of the polypeptide chain: α-helices, β-strands, and loops. Groups of secondary structures usually combine to form compact globular structures, which represent the three-dimensional tertiary structure of an entire protein. The central dogma of protein science is that the primary structure determines the tertiary structure. Although this is not necessarily true in all cases (e.g., some proteins require chaperone proteins to facilitate their folding process), this dogma is tacitly assumed for most of the computational techniques used for predicting and comparing the structure of globular proteins.

In the following sections we consider problems related to the prediction and comparison of protein structures. Knowing the structure of a protein provides a basis for identifying the protein's function, and protein structures are necessary for many computational drug docking techniques. (Although it is an important research topic, we shall not explore drug docking problems here, except to say that it is the binding of a small molecule, called a *ligand*, like a drug, to some site on a protein. There are many interesting open optimization problems, such as the location of the docking site, which are described in Nussinov et al. 2002.) Because of its importance, a variety of techniques have been developed to predict protein structure, but only a few are based on mathematical programming. We summarize efforts to characterize the computational complexity of protein structure prediction, and we describe one approach in detail.

Protein-alignment techniques can be used to compare and predict protein structures. We describe recent work on protein alignment with contact maps, which can be applied to assess the accuracy of protein structure prediction methods (given known protein structures). For a sense of scale, the number of amino acid residues in a protein ranges from about 50 to 2500. As a practical matter, instances in the range 100–500 are interesting enough to be studied because we do not understand the function of many of those proteins.

5.1. Protein Structure Prediction

One approach to protein structure prediction is to determine the position of a protein's atoms so as to minimize the total free energy (Byrd et al. 1996, Neumaier 1997). In practice, accurate energy function calculations cannot be used for protein structure prediction, even for proteins with as few as 50 residues, so approximate models are commonly used. Even these simplified models are complex, and they frequently have many local minima. Here we consider a lattice model of protein folding based on the following biological simplification due to Dill (1985) (also see Lau and Dill 1989, Dill et al. 1995). Amino acids can be hydrophobic, which means that they do not do well in water, or hydrophilic, which means that they do. Using this simplification, optimization models have been developed during the past decade that seek to maximize interactions between adjacent pairs of hydrophobic side chains. The adjacency is defined on a lattice of points that can be regarded here as a discrete approximation, or grid, in space. The rationale for this objective is that hydrophobic interactions contribute a significant portion of the total energy function. Roughly, this objective favors conformations that have the hydrophobic amino acid residues clustered on the inside, covered by the hydrophilic ones.

Lattice models of protein folding have provided valuable insights into the general complexity of protein structure prediction problems. For example, protein structure prediction has been shown to be NP-hard for a variety of lattice models (Atkins and Hart 1999, Berger and Leighton 1998, Crescenzi et al. 1998, Hart and Istrail 1997b). This lends credibility to the general assumption that protein structure prediction is an intractable problem. These results are complemented by performance-guaranteed approximation algorithms that run in linear time. These results show that near-optimal protein structures can be quickly constructed, and they can be generalized to simple off-lattice protein models (Hart and Istrail 1997a).

Most of the complexity analysis and algorithm design has focused on variations of Dill's hydrophobic-hydrophilic model. This is generally called the *HP model*, where P is used to denote hydrophilic because those amino acids are also polar. The HP model is one of the most studied simple (globular) protein models. From a computational view, we are reducing the alphabet from 20 characters to two, where our input sequences are from $\{H, P\}^+$. The recent results of Andorf et al. (2002) explore the range of alphabet size, from 2 to 20, taking other properties of amino acids into consideration.

In addition to providing insight into the theoretical computational complexity of protein structure prediction, the HP model has also been used to assess and evaluate many different optimization techniques applied to this problem. A wide range of heuristics have been applied to find optimal HP structures, especially evolutionary algorithms (Khimasia and Coveney 1997; Krasnogor et al. 1998a, b; Patton et al. 1995; Piccolboni and Mauri 1998; Rabow and Scheraga 1996; Unger and Moult 1993a, b). Additionally, exact methods for protein structure prediction in the HP model have been developed using constrained enumeration techniques (Yue and Dill 1993, 1994) and logic programming (Backofen 1999). Solving HP problems with either heuristic or exact methods has proven quite challenging, and none of these

methods is able to scale robustly to sequences of hundreds of amino acids.

Thus, there appears to be an opportunity for mathematical programming techniques to obtain deeper insight into these problems. These insights can be biological, or they can be exploited algorithmically to solve problems of practical size. We begin with a simple ILP model.

A residue's neighbor (successor or predecessor in the sequence) must be assigned to a neighboring grid point (exactly one unit away). Mathematically, if the assigned coordinates of two neighbors of the sequence are (x_i, y_i, z_i) and $(x_{i+1}, y_{i+1}, z_{i+1})$, this must satisfy:

$$|x_i - x_{i+1}| + |y_i - y_{i+1}| + |z_i - z_{i+1}| = 1.$$

The 2D counterpart is easier to visualize. Figure 8(a) shows a grid for nine acids and the backbone of a protein. The open circles are hydrophilic acids, and the filled circles are hydrophobic acids. In this state, the number of hydrophobic contacts is three: (1,2), (5,6), and (6,7) are hydrophobic neighbors. Figure 8(b) shows a fold, adding two to its number of hydrophobic contacts, shown by the dotted line connecting them. (The hydrophobic contacts along the backbone are often omitted in HP models, because any conformation always includes these contacts. However, we include them here to simplify the statement of our ILP model.)

The amino acid residue sequence is denoted $\langle a_1, \ldots, a_n \rangle$, and the set of lattice points is $\mathcal{L} = \{1, 2, \ldots, n^2\}$, such that the coordinates are of the form:

$$y_p = \left\lfloor \frac{p-1}{n} \right\rfloor \quad \text{and} \quad x_p = p - 1 - ny_p \quad \text{for } p \in \mathcal{L}.$$

Define the neighborhood of a point by

$$\mathcal{N}(p) = \{q \in \mathcal{L}\colon |x_p - x_q| + |y_p - y_q| = 1\}.$$

Let v_{ip} be a binary decision variable with the following meaning:

$$v_{ip} = \begin{cases} 1 & \text{if acid } a_i \text{ is assigned to point } p, \\ 0 & \text{otherwise.} \end{cases}$$

The following constraints give us the correspondence we seek.

(I) Every residue must be assigned to a point:

$$\sum_{p \in \mathcal{L}} v_{ip} = 1 \quad \text{for } i = 1, \ldots, n.$$

(II) No point can be assigned more than one residue:

$$\sum_{i=1}^{n} v_{ip} \le 1 \quad \text{for } p \in \mathcal{L}.$$

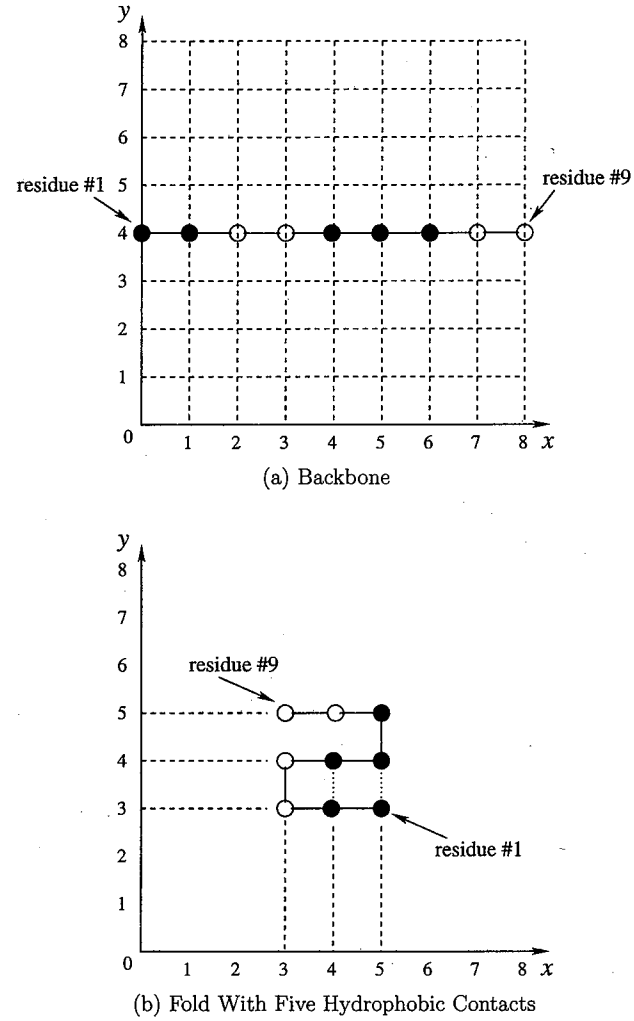

Figure 8 **Protein in a 9×9 Lattice**

(III) Sequence order must be preserved:

$$\sum_{q \in \mathcal{N}(p)} v_{i+1,q} \ge v_{ip} \quad \text{for } i = 1, \ldots, n-1,\ p \in \mathcal{L}$$

$$\sum_{q \in \mathcal{N}(p)} v_{i-1,q} \ge v_{ip} \quad \text{for } i = 2, \ldots, n,\ p \in \mathcal{L}.$$

Now we need to count the number of hydrophobic contacts for any assignment, v, with binary hydrophobic contact variables. Define

$$h_{pq} = \begin{cases} 1 & \text{if there are hydrophobic residues} \\ & \quad \text{assigned to } p \text{ and } q, \\ 0 & \text{otherwise,} \end{cases}$$

for $(p, q) \in \mathcal{D} \stackrel{\text{def}}{=} \{(p, q)\colon q \in \mathcal{N}(p)\}$ (=domain of h).

The following constraints provide part of the assurance that h is determined appropriately.

(IV)

$$h_{pq} \le \sum_{i \in \mathcal{H}} v_{ip} \quad \text{and} \quad h_{pq} \le \sum_{i \in \mathcal{H}} v_{iq} \text{ for } (p, q) \in \mathcal{D},$$

where $\mathcal{H}$ is the set of indices of hydrophobic acids. Each sum on the right is zero or one for any assignment, v. If either is zero, that forces $h_{pq}=0$, so we do not allow this to "score." That is what we want: *do not score any hydrophobic contact if the neighboring points are not both hydrophobic.* If both right-hand sides are one, that means $v_{ip}=1$ for some residue i that is hydrophobic; and, $v_{kq}=1$ for some other residue k that is also hydrophobic. That allows $h_{pq}=1$.

The complete ILP model is given by the following:

$$\max \sum_{(p,q)\in\mathcal{D}} h_{pq}\text{: (I)–(IV) and } h_{pq}, v_{ip} \in \{0,1\}\,\forall i,p,q.$$

The maximization takes care of determining the hydrophobic contact scoring variables (h) correctly. If the assignment allows $h_{pq}=1$, that is the value it will be in an optimal solution. Although this model is correct, we consider several ways in which it can be improved.

We must eliminate translation, rotation, and reflection symmetries because branch-and-bound does not understand and exploit the geometry of this problem. It will see a symmetric solution as an alternative optimum, even though it is the same fold. This means that the search tree will be huge because it will include paths to the symmetric solutions that it cannot close. Fortunately, there is a simple fix to avoid translation symmetries: assign some acid, say m, to some point, say p_m. (We have found it effective to fix the middle acid to the middle of the lattice.) To do this, we simply add the constraint $v_{mp_m}=1$.

The complete lattice is now reduced by half because not every point is reachable. For example, $a_{m\pm1}$ must be at one of the four points neighboring p_m. Letting $\mathcal{L}_R$ denote the set of reachable points, this reduces the neighborhood to $\mathcal{N}(p)=\{q\in\mathcal{L}_R\colon |x_p-x_q|+|y_p-y_q|\le 1\}$. This carries to the domain of h: $\mathcal{D}=\{(p,q)\colon p\in\mathcal{L}_R,\ q\in\mathcal{N}(p)\}$. We point this out because other domain reductions are possible, and we continue to use $\mathcal{N}(p)$ and $\mathcal{D}$ with the understanding that they are the reduced sets.

We now consider the effect of rotation and reflection symmetries. Note that we can rotate any fold about the middle to put a_1 into some quadrant, say quadrant 3. Then, we can reflect this rotated fold about the line $y=x$ to put a_1 into the half-quadrant. The following constraint eliminates these symmetries:

$$\sum_{\substack{p\colon x_p\le x_{p_m}\\ y_p\le y_{p_m}\\ y_p\ge x_p}} v_{1p}=1.$$

This region is illustrated in Figure 9. Not all symmetries can be eliminated at the outset, and Backofen (1999) presents rules that deal with new symmetries that arise during branch and bound.

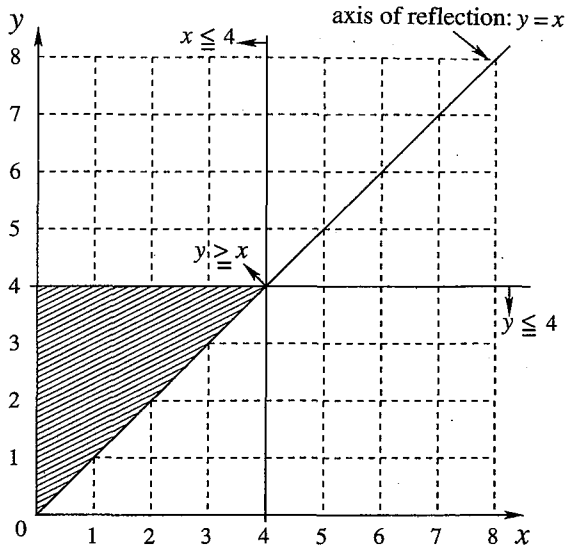

Figure 9 **Region Restricting a_1 to Eliminate Some Symmetric Folds**

We now consider a different formulation by adding new variables:

$$E_{ipq}=\begin{cases}1 & \text{if } v_{ip}=v_{i+1,q}=1,\\ 0 & \text{otherwise}\end{cases}$$

for $i=1,\ldots,n-1$, $(p,q)\in\mathcal{D}$. In words, E describes a pair of assignments: Its value is 1 if, and only if, a_i is assigned to point p and its successor, a_{i+1}, is assigned to the neighboring point, q. A potential improvement comes from reformulating the sequence-preserving constraints.

It is useful to think of the lattice as a network whose nodes are the reachable points and whose edges are links with neighbors. Then, we can think of E_{ipq} as flow from i to $i+1$ across the edge (p,q), as depicted in Figure 10. We can relate the assignment variables to these flow variables as follows:

$$\text{outflow: } v_{ip}=\sum_{q\in\mathcal{N}(p)} E_{ipq} \quad \text{for } i=1,\ldots,n-1,\ p\in\mathcal{L}_R$$

$$\text{inflow: } v_{ip}=\sum_{q\in\mathcal{N}(p)} E_{i-1,qp} \quad \text{for } i=2,\ldots,n,\ p\in\mathcal{L}_R$$

$$\text{backflow: } v_{ip}\ge E_{ipq}+E_{i-1,qp} \quad \text{for } i=2,\ldots,n,\ (p,q)\in\mathcal{D}.$$

The outflow constraint says that if a_i is assigned to point p ($v_{ip}=1$), its successor must be assigned to some neighbor. Conversely, if a_i is not assigned to point p ($v_{ip}=0$), all of the associated flow variables must be zero. The inflow constraint says that if a_i is assigned to point p, its predecessor (a_{i-1}) must be assigned to some reachable neighbor. Conversely, if a_i is not assigned to point p, all of the associated flow variables must be zero. The backflow constraint says that if a_i is assigned to point p, either its successor or predecessor is assigned to a neighbor (q), but not

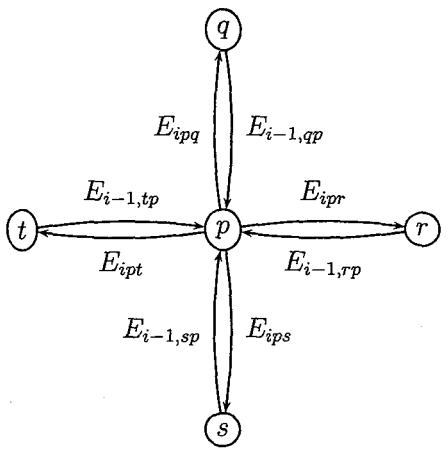

Figure 10 **Flow Variables (E_{ipq}) Added to Model**

both (to the same neighbor). Conversely, if a_i is not assigned to point p, the two flow variables are forced to be zero (but this is not an additional constraint since the outflow and inflow constraints force this).

THEOREM 5.1. *If (v, E) satisfies the flow constraints, v satisfies the sequence-order-preserving constraints.*

PROOF. Here the "sequence-order-preserving constraints" are (III). The backflow and outflow constraints yield:

$$\sum_{q\in\mathcal{N}(p)} v_{i+1,q} \geq \sum_{q\in\mathcal{N}(p)} (E_{i+1qp} + E_{ipq}) \geq \sum_{q\in\mathcal{N}(p)} E_{ipq} = v_{ip}$$

for $i = 1, \ldots, n-1$. Similarly, the backflow and inflow constraints yield

$$\sum_{q\in\mathcal{N}(p)} v_{i-1,q} \geq \sum_{q\in\mathcal{N}(p)} (E_{i-1,qp} + E_{i-2,pq}) \geq \sum_{q\in\mathcal{N}(p)} E_{i-1,qp} = v_{ip}$$

for $i = 2, \ldots, n$. □

The flow constraints ensure that the backbone sequence is preserved, so we drop the original order-preserving constraints (III).

The complete, revised formulation is

$$\text{IP}': \quad \max \sum_{p\in\mathcal{L}_R} \sum_{q\in\mathcal{N}(p)} h_{pq}\colon v_{ip}, \qquad E_{ipq} \in \{0, 1\},$$
$$0 \leq h_{pq} \leq 1,$$

$$\text{Assignment:} \quad \sum_p v_{ip} = 1, \qquad \sum_i v_{ip} \leq 1,$$

$$\text{Flow:} \quad v_{ip} = \sum_{q\in\mathcal{N}(p)} E_{ipq} = \sum_{q\in\mathcal{N}(p)} E_{i-1,qp}, \quad v_{ip} \geq E_{ipq} + E_{i-1,qp},$$

$$\text{Scoring limits:} \quad h_{pq} \leq \sum_{i\in\mathcal{H}} v_{ip}, \quad h_{pq} \leq \sum_{i\in\mathcal{H}} v_{iq},$$

$$\text{Restrict } a_1\text{:} \quad \sum_{p\in Q} v_{1p} = 1 \ (Q \text{ is } 1/2 \text{ quadrant } 3),$$

$$\text{Fix middle:} \quad v_{mp_m} = 1.$$

(Domain restrictions and boundary conditions are as in the original IP.)

Theorem 5.1 shows that every 0–1 solution to IP′ corresponds to a 0–1 solution to IP. It is not difficult to prove the converse, so IP and IP′ have the same space of 0–1 assignments and yield the same objective value over those assignments. Theorem 5.1 suggests that IP′ is sharper than IP, but we think that the LP relaxation of IP′ has many more extreme points, so whether this flow formulation is computationally better than the simpler model (IP) requires further study.

5.2. Contact Map Alignment

One approach to understanding a new protein's function is to see if it is similar to some known protein. One measure of similarity, which is described here, uses knowledge of the structure of both proteins (e.g., the structures of known native states).

The *contact map* of a protein is a graph with a node for each amino acid residue and an edge for each pair of non-adjacent residues whose distance is within a given threshold. The distance between two residues can be defined, for example, as the smallest Euclidean distance between any pair of atoms in the residues. Given contact maps for two proteins, $G_1 = [V_1, E_1]$ and $G_2 = [V_2, E_2]$, a similarity measure is the relative size of the maximal subgraphs that are isomorphic while preserving their backbone sequences (Goldman et al. 1999).

We illustrate this similarity metric in Figure 11. The first protein has eight residues and the second protein has ten. The alignment shows the residues selected in the subgraphs (nodes 1, 2, 4, 5, 6, 7, and 8 from V_1, and nodes 1, 2, 3, 5, 7, 9, and 10 from V_2). The linear order is preserved by associating 1–1, 2–2, 4–3, 5–5, 6–7, 7–9, and 8–10, and the dark edges illustrate the isomorphic edges in each contact map. These edges satisfy the condition that their endpoints are associated. For example, the edge (1, 4) in E_1 corresponds to edge (1, 3) in E_2 because of the node associations 1–1 and 4–3.

Now let us formulate the contact map optimization (CMO) problem as a 0–1 IP. We define $x_{ij} = 1$ iff $i \in V_1$ is associated with $j \in V_2$, and we define $y_{(i,k)(j,l)} = 1$ iff edges $(i, k) \in E_1$ and $(j, l) \in E_2$ are selected. The objective of CMO is to maximize $\sum y_{(i,k)(j,l)}$.

The selected edges must have their endpoints associated, so

$$y_{(i,k)(j,l)} = 1 \implies x_{ij} = x_{kl} = 1,$$

which we express with the inequalities:

$$y_{(i,k)(j,l)} \leq x_{ij} \quad \text{and} \quad y_{(i,k)(j,l)} \leq x_{kl} \qquad \text{for } (i,k) \in E_1, \quad (j,l) \in E_2.$$

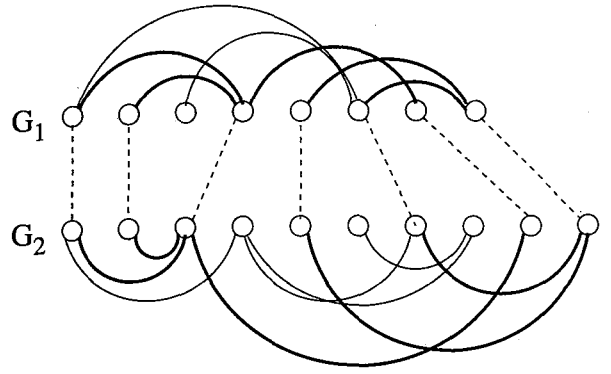

Figure 11 **Example Contact Map Alignment With Isomorphic Subgraphs With Five Edges (Lancia et al. 2001b)**

The x_{ij} associations must be unique, so

$$\sum_{i\in V_1} x_{ij} \le 1,\ \forall j \in V_2 \quad \text{and} \quad \sum_{j\in V_2} x_{ij} \le 1,\ \forall i \in V_1.$$

These are the usual assignment limits: At most one node in one graph can be associated with a node in the other graph. Finally, at most one of the two associations that cross is allowed, so we have

$$x_{ij}+x_{kl}\le 1 \quad \text{for} \quad 1\le i<k\le |V_1| \quad \text{and} \quad 1\le l<j\le |V_2|.$$

Here is the complete formulation:

$$\max \sum_{\substack{(i,k)\in E_1\\(j,l)\in E_2}} y_{(i,k)(j,l)}\colon x_{ij}, y_{(i,k)(j,l)} \in \{0,1\},$$

$$y_{(i,k)(j,l)} \le x_{ij} \quad \text{and} \quad y_{(i,k)(j,l)} \le x_{kl} \text{ for } (i,k)\in E_1,\ (j,l)\in E_2$$

$$\sum_{i\in V_1} x_{ij} \le 1 \quad \text{for } j\in V_2,$$

$$\sum_{j\in V_2} x_{ij} \le 1 \quad \text{for } i\in V_1,$$

$$x_{ij}+x_{kl}\le 1 \quad \text{for} \quad 1\le i<k\le |V_1| \quad \text{and} \quad 1\le l<j\le |V_2|.$$

This formulation can be decomposed into x and y variables, from which we derive valid inequalities on the x problem. Define

$$Y(x) = \{(i,j,k,l)\colon (i,k)\in E_1,\ (j,l)\in E_2, \text{ and } x_{ij}=x_{kl}=1\}.$$

Then, for any (fixed) x, the optimal choice of y is given by:

$$y^*_{(i,k)(j,l)}(x) = \begin{cases} 1 & \text{if } (i,j,k,l)\in Y(x),\\ 0 & \text{otherwise.}\end{cases}$$

Thus, we can reformulate the CMO problem as

$$\max |Y(x)|\colon$$

$$x_{ij}\in\{0,1\}, \quad \sum_i x_{ij}\le 1, \quad \sum_j x_{ij}\le 1, \quad x_{ij}+x_{kl}\le 1.$$

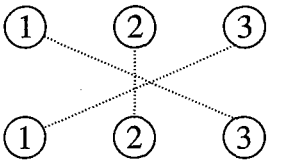

Figure 12 **Three Associations That Are Pairwise Crossings**

The constraints in this formulation define an independent set in a related graph.

This observation can be used to strengthen the IP for CMO. In particular, if we have three or more pairwise crossing associations, their exclusion inequalities can be strengthened. For example, the associations shown in Figure 12 give the inequalities

$$x_{13}+x_{22}\le 1, \quad x_{13}+x_{31}\le 1, \quad x_{22}+x_{31}\le 1.$$

However, these can be replaced by the stronger inequality:

$$x_{13}+x_{22}+x_{31}\le 1.$$

Both systems have the same 0–1 solutions: At most one of the three associations can be made, to the exclusion of the other two. However, the one inequality is stronger because it has a smaller set of fractional solutions. In particular, the first system admits $(1/2, 1/2, 1/2)$, which violates the second system.

The linear programming relaxation (LPR) of this problem has some surprises. Figure 13 shows two contact maps with six residues. The graphs are isomorphic, so the optimal alignment selects all eight pairs of edges. One might think the LPR would be integer-valued, but the opposite is true.

The fractional solution to this example has $x_{ij}=1/6$ for all values, except $x_{22}=x_{66}=1/3$. These assignments give nearly the least information possible by having the fractional values be nearly equal. The crossing constraints are satisfied because they admit any pair of assignments whose sum does not exceed one; in fact, none of the crossing constraints is binding in the fractional solution.

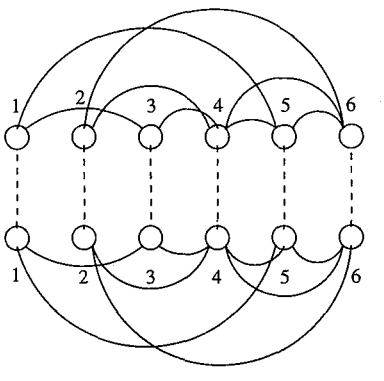

Figure 13 **Isomorphic Contact Maps**

The solution does not change if any 3-, 4-, 5-, or 6-cliques are added because their sums are still less than one. We can, however, extend the clique inequalities beyond six variables by including touching as a crossing. Start with the 6-clique inequality:

$$x_{16} + x_{25} + x_{34} + x_{43} + x_{52} + x_{61} \leq 1.$$

We can add x_{15} to this because it crosses all lines and touches x_{16} (which is a violation). In fact, we have the following maximal, extended clique inequality:

$$x_{16} + x_{25} + x_{34} + x_{43} + x_{52} + x_{61} + x_{15} + x_{24} + x_{33} + x_{42} + x_{51} \leq 1.$$

This eliminates the current fractional solution. The technique for finding this maximal inequality is given by Lancia et al. (2001b).

6. Epilogue

Most of the problems we described are NP-hard, so it is unlikely that anyone will produce a practical algorithm that guarantees an optimal solution within a realistic amount of time. Still, exact methods are important because (a) for some problems they are applicable to practical problem instances, and (b) they can be used to benchmark fast heuristics on specific problems. The multiple sequence alignment problem, for example, has polynomial complexity when the number of sequences to be aligned is fixed. More importantly, new knowledge of the underlying science can reveal that the dimension of a problem is much less than we are using. For example, protein folding might depend on key subsequences of amino acids, rather than on their total number. Thus, research is needed to improve general branch-and-bound or dynamic programming methods by exploiting special problem structure. One must also be mindful of the underlying biology, questioning whether we are measuring complexity by the right dimension. This underlies the Levinthal Paradox (Ngo et al. 1994), which discusses these points in greater detail. The "paradox" is that nature solves problems, like protein folding, in seconds (or less), but our fastest computers cannot. This raises questions about how nature works; it is unlikely that nature solves the same problem we pose, and the in-depth discussion provided by Ngo et al. (1994) is important to read.

When the dimensionality is too great to guarantee an exact optimal solution, there are two approaches: (1) metaheuristics and (2) approximation algorithms. The former is a skillfully guided set of rules, often based on some metaphor of nature; the latter gives a guarantee of some percentage of optimality. Both have been applied to most of the problems discussed in this paper, but much more is needed. There are many opportunities for research into the design and analysis of computational methods familiar to researchers in mathematical programming, coming from an operations research background. Even less has been done with sensitivity analysis and optimizing under uncertainty.

Finally, we remind the reader that the solution to these types of combinatorial problems does not necessarily mean that the related biological problem is solved. In our examples, the mathematical model defines an abstraction that emphasizes some particular aspects of the underlying biological system. For example, many different protein structure prediction problems have been formulated, each of which emphasizes different aspects of the protein folding process with different levels of fidelity (e.g., using a simple hydrophobic model versus a detailed model based on the statistical mechanics of water). Consequently, the solution to a combinatorial optimization problem often leads to stimulating discussions with biologists regarding how these results can be interpreted within a biological context. Further, these discussions often generate new ideas for combinatorial formulations that might provide further insight into the underlying problem. In our opinion, this dynamic interaction with biologists is one of the most productive aspects of this research area, making it also one of the most exciting.

Appendix: Getting Started

Some light reading about computational biology is provided by Karp (2002). A good place to begin a more thorough study is with Ph.D. theses. These have been carefully reviewed, and the authors have spent a good deal of time providing background, perspective, and complete references. The authors recommend the theses of Backofen (1999), Lancia (1997), and Pedersen (1999).

For a gentle introduction to the biology, see Brown (1999), Hunter (1993). Then, the authors recommend studying Campbell and Heyer (2002). The number of books on computational molecular biology has increased since the early entry by Setubal and Meidanis (1997). For the kinds of problems described here the authors recommend Pevzner (2000) and Clote and Backofen (2000).

The primary journals and proceedings in this field are as follows (†indicates free of cost):

- *Bioinformatics* http://bioinformatics.oupjournals.org/
- †*In Silico Biology* http://www.bioinfo.de/isb/
- *Journal of Computational Biology* http://www.liebertpub.com/CMB/default1.asp
- †*Proceedings of the Pacific Symposium on Biocomputing* http://psb.stanford.edu/
- *Research in Computational Biology Proceedings* (RECOMB) http://www.recomb2003.de/
- *Proceedings of the Intelligent Systems for Molecular Biology* (ISMB) http://www.iscb.org/ismb2003/.

Links to more journals and resources are at the *International Society for Computational Biology*, http://www.iscb.org/, which also contains links to their symposia proceedings.

The web also contains a wealth of information about computational biology, molecular biology, and related topics. Searching for courses and primers yields greater results each year, as universities are presenting preparatory courses in the relevant elements of biosciences for the computer scientist, engineer, or mathematician. Here are some suggestions for getting started.

- *Biology Hypertextbook*, from MIT (Dakla et al. 2004)

This is an ongoing project that started in 1996 and has been developed by many MIT faculty and students. It is divided into 11 chapters, starting with a chemistry review and ending with immunology. If you search the index, you will find some untitled modules that indicate work in progress.

- *Molecular Biology Notebook*, from Rothamsted Research (Molecular Biology Notebook 2000)

This contains many online short courses, starting with *The Beginner's Guide to Molecular Biology*, which contains 13 lessons, beginning with a definition of life and ending with molecular engineering.

Acknowledgments

This work was performed in part at Sandia National Laboratories. Sandia is a multipurpose laboratory operated by Sandia Corporation, a Lockheed-Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000. This work was partially funded by the U.S. Department of Energy's *Genomes to Life* program (http://www.doegenomestolife.org), under project "Carbon Sequestration in Synechococcus Sp.: From Molecular Machines to Hierarchical Modeling," (http://www.genomes-to-life.org). Harvey Greenberg thanks Sandia National Laboratories for the opportunity to visit their Computing Research Institute in 2000, which is where the author was introduced to computational biology. The authors thank Alberto Caprara for suggesting the simple proof of Theorem 3.1 and Graziano Pesole for the model presented in Section 3.4. The authors thank Naiomi Cameron, Robert Carr, Jonathan Eckstein, Sorin Istrail, Alantha Newman, and Cynthia Phillips for their collaboration developing IP formulations for protein folding in lattice models (Section 5.1). Last, the authors thank three anonymous referees, as well as Courtney Davis, Allen G. Holder, and Ed Wasil, for detailed comments and suggestions that made this paper clearer.

References

Andorf, C., D. Dobbs, V. Honavar. 2002. Discovering protein function classification rules from reduced alphabet representations of protein sequences. Technical report, Department of Computer Science, Iowa State University, Ames, IA.

Atkins, J., W. E. Hart. 1999. On the intractability of protein folding with a finite alphabet of amino acids. *Algorithmica* **25** 279–294.

Ausiello, G., P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, M. Protasi. 1999. *Complexity and Approximation. Combinatorial Optimization Problems and their Approximability Properties*. Springer, Berlin, Germany.

Babai, L. 1991. Local expansion of vertex-transitive graphs and random generation in finite groups. *Proc. ACM Sympos. Theory Comput. (STOC)*, ACM Press, New York, 164–174.

Backofen, R. 1999. *Optimization Techniques for the Protein Structure Prediction Problem*. Ph.D. thesis, Ludwig-Maximilians-Universität München, Institut für Informatik, München, Germany.

Bader, D. A., B. M. Moret, M. Yan. 2001. A linear-time algorithm for computing inversion distances between signed permutations with an experimental study. *J. Comput. Biol.* **8**(5) 483–491.

Bafna, V., P. Pevzner. 1996. Genome rearrangements and sorting by reversals. *SIAM J. Comput.* **25** 272–289.

Bafna, V., P. Pevzner. 1998. Sorting by transpositions. *SIAM J. Discrete Math.* **11**(2) 224–240.

Bafna, V., E. L. Lawler, P. Pevzner. 1997. Approximation algorithms for multiple sequence alignment. *Theoret. Comput. Sci.* **182**(1–2) 233–244.

Berger, B., T. Leighton. 1998. Protein folding in the hydrophobic-hydrophilic (HP) model is NP-complete. *J. Comput. Biol.* **5**(1) 27–40.

Berman, P., M. Karpinski. 1999. On some tighter inapproximability results. J. Wiederman, P. van Emde Boas, M. Nielsen, eds. *Automata, Languages and Programming, 26th International Colloquium, ICALP'99*, No. 1644. *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany.

Berman, P., S. Hannenhalli, M. Karpinski. 2002. 1.375-approximation algorithm sorting by reversals. *Proc. Annual Eur. Sympos. on Algorithms (ESA)*, No. 2461. *Lecture Notes in Computer Science*, Springer, Heidelberg, Germany, 200–210.

Berman, H., J. Westbrook, Z. Feng, G. Gilliland, T. Bhat, H. Weissig, I. Shindyalov, P. Bourne. 2000. The protein data bank. *Nucleic Acids Research* **28** 235–242. The PDB is at http://www.rcsb.org/pdb/.

Błażewicz, J., M. Kasprzak, M. Stema, J. Weglarz. 1997. Selected combinatorial optimization problems arising in molecular biology. *Ricerca Operativa* **26**(80) 35–63.

Bower, J., H. Bolouri, eds. 2001. *Computational Modeling of Genetic and Biochemical Networks*. MIT Press, Cambridge, MA.

Brown, T. 1999. *Genomes*. John Wiley & Sons, New York.

Byrd, R., E. Eskow, A. van der Hoek, R. Schnabel. 1996. Global optimization methods for protein folding problems. P. Pardalos, D. Shalloway, G. Xue, eds. *Proc. DIMACS Workshop on Global Minimization Nonconvex Energy Functions: Molecular Conformation and Protein Folding, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, American Mathematical Society, Providence, RI, 29–40.

Campbell, A., L. Heyer. 2002. *Discovering Genomics, Proteomics, & Bioinformatics*. Benjamin Cummings, San Francisco, CA.

Caprara, A. 1999a. On the tightness of the alternating-cycle lower bound for sorting by reversals. *J. Combin. Optim.* **3** 149–182.

Caprara, A. 1999b. Sorting permutations by reversals and Eulerian cycle decompositions. *SIAM J. Discrete Math.* **12** 91–110.

Caprara, A., G. Lancia. 2000. Experimental and statistical analysis of sorting by reversals. D. Sankoff, J. H. Nadeau, eds. *Comparative Genomics: Empirical and Analytical Approaches to Gene Order Dynamics, Map Alignment and Evolution of Gene Families*, No. 1. Kluwer Series in Computational Biology, Kluwer, Norwell, MA, 171–183.

Caprara, A., G. Lancia, S. Ng. 2001. Sorting permutations by reversals through branch and price. *INFORMS J. Comput.* **13**(3) 224–244.

Carrillo, H., D. Lipman. 1988. The multiple sequence alignment problem in biology. *SIAM J. Appl. Math.* **48**(5) 1073–1082.

Clark, A. 1990. Inference of haplotypes from PCR—amplified samples of diploid populations. *Mol. Biol. Evolution* **7** 111–122.

Clote, P., R. Backofen. 2000. *Computational Molecular Biology*. John Wiley & Sons, New York.

Crescenzi, P., D. Goldman, C. Papadimitriou, A. Piccolboni, M. Yannakakis. 1998. On the complexity of protein folding. *J. Comput. Biol.* **5**(3) 423–466.

Dakla, E., A. Latham, E. Tecle, G. Tsan, L. Willis, eds. 2004. *MIT Biology Hypertextbook*. Massachusetts Institute of Technology, http://web.mit.edu/esgbio/www/.

Delcher, A., S. Kasif, R. Fleischmann, J. Peterson, O. White, S. Salzberg. 1999. Fast algorithms for whole genomes. *Nucleic Acids Res.* **27**(11) 2369–2376. Also see http://www.tigr.org/software/mummer/.

Delcher, A., A. Phillippy, J. Carleton, S. Salzberg. 2002. Fast algorithms for whole genomes. *Nucleic Acids Res.* **30**(11) 2478–2483.

Dill, K. A. 1985. Theory for the folding and stability of globular proteins. *Biochemistry* **24** 1501.

Dill, K. A., S. Bromberg, K. Yue, K. Fiebig, D. Yee, P. Thomas, H. Chan. 1995. Principles of protein folding—A perspective from simple exact models. *Protein Sci.* **4** 561–602.

Fuellen, G. 1997. *A Gentle Guide to Multiple Alignment*. World Wide Web, http://www.techfak.uni-bielefeld.de/bcd/Curric/MulAli/mulali.html.

Garey, M., D. Johnson. 1979. *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W.H. Freeman and Co., San Francisco, CA.

Genome. 2001. The Genome International Sequencing Consortium, Initial sequencing and analysis of the human genome. *Nature* **409** 860–921.

Goldman, D., S. Istrail, C. Papadimitriou. 1999. Algorithmic aspects of protein structure similarity. *40th Annual Sympos. Foundations Comput. Sci. (FOCS)*. IEEE Computer Society Press, New York, 512–521.

Golumbic, M. 1980. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York.

Greenberg, H. 1996–2003. *Mathematical Programming Glossary*. World Wide Web, http://www.cudenver.edu/~hgreenbe/glossary/.

Greenberg, H., R. Bar-Or, L. Lana, C. Miller, T. Morrison, C. van Woudenberg. 2002. Pathway inference: Computational issues. Mathematics clinic report, Mathematics Department, University of Colorado, Denver, CO. Available at http://www-math.cudenver.edu/clinic/report.shtml.

Groetschel, M., L. Lovasz, A. Schrijver. 1984. A polynomial algorithm for perfect graphs. *Ann. Discrete Math.* **21** 325–356.

Gusfield, D. 1993. Efficient methods for multiple sequence alignment with guaranteed error bounds. *Bull. Math. Biol.* **55** 141–154.

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

Gusfield, D. 1999. Inference of haplotypes from PCR—amplified samples of diploid populations: Complexity and algorithms. Technical report CSE-99-6, University of California, Department of Computer Science, Davis, CA.

Gusfield, D. 2000. A practical algorithm for optimal inference of haplotypes from diploid populations. R. Altman, T. Bailey, P. Bourne, M. Gribskov, T. Lengauer, I. Shindyalov, L. T. Eyck, H. Weissig, eds. *Proc. Eighth Internat. Conf. Intelligent Systems Molecular Biol. (ISMB)*, AAAI Press, Menlo Park, CA, 183–189.

Gusfield, D. 2001. XPARAL: Graphical computation of parameterized alignments. http://www.cs.ucdavis.edu/~gusfield/xparall/.

Gusfield, D. 2003. Haplotyping by pure parsimony. Technical report CSE-2003-2, Department of Computer Science, University of California, Davis, CA.

Hannenhalli, S., P. A. Pevzner. 1995 (full version appeared in *Journal of the ACM* 46, 1–27, 1999). Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals). *Proc. ACM Sympos. Theory Comput. (STOC)*, ACM Press, New York, 178–189.

Hart, W. E., S. Istrail. 1997a. Lattice and off-lattice side chain models of protein folding: Linear time structure prediction better than 86% of optimal. *J. Comput. Biol.* **4**(3) 241–259.

Hart, W. E., S. Istrail. 1997b. Robust proofs of NP-hardness for protein folding: General lattices and energy potentials. *J. Comput. Biol.* **4**(1) 1–20.

Hunter, L. 1993. Molecular biology for computer scientists. L. Hunter, ed. *Artificial Intelligence & Molecular Biology*. MIT Press, Cambridge, MA, 1–46. Available at http://www.aaai.org/Library/Books/Hunter/hunter.html.

Ideker, T., T. Galitski, L. Hood. 2001. A new approach to decoding life: Systems biology. *Annual Rev. Genomics Human Genetics* **2** 343–372.

Jiang, T., E. Lawler, L. Wang. 1994. Aligning sequences via an evolutionary tree: Complexity and approximation. *Proc. Annual ACM Sympos. Theory Comput. (STOC)*. ACM Press, New York, 760–769.

Kaplan, H., R. Shamir, R. E. Tarjan. 1997. Faster and simpler algorithm for sorting signed permutations by reversals. *Proc. Annual ACM-SIAM Sympos. Discrete Algorithms (SODA)*. ACM Press, New York, 344–351.

Karp, R. 2002. Mathematical challenges from genomics and molecular biology. *Notices Amer. Math. Soc.* **49**(5) 544–553.

Kececioglu, J., R. Ravi. 1995. Of mice and men: Algorithms for evolutionary distances between genomes with translocation. *Proc. Annual ACM-SIAM Sympos. Discrete Algorithms (SODA)*, ACM Press, New York, 604–613.

Kececioglu, J., D. Sankoff. 1995. Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement. *Algorithmica* **13** 180–210.

Khimasia, M., P. Coveney. 1997. Protein structure prediction as a hard optimization problem: The genetic algorithm approach. *Molecular Simulation* **19** 205–226.

Krasnogor, N., P. M. López, E. de la Canal, D. Pelta. 1998a. Simple models of protein folding and a memetic crossover. *Exposed at INFORMS CSTS, Computer Science and Operations Research: Recent Advances in the Interface Meeting*.

Krasnogor, N., D. Pelta, P. M. Lopez, P. Mocciola, E. de la Canal. 1998b. Genetic algorithms for the protein folding problem: A critical view. E. Alpaydin, C. Fyfe, eds. *Proc. Engrg. Intelligent Systems*, Academic Press, Reading, MA.

Kurtz, S. 1999. Reducing the space requirement of suffix trees. *Software-Practice Experience* **29**(13) 1149–1171.

Lancia, G. 1997. *Optimization Problems in Computational Molecular Biology*. Ph.D. thesis, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA.

Lancia, G., V. Bafna, S. Istrail, R. Lippert, R. Schwartz. 2001a. SNPs problems, complexity and algorithms. *Proc. Annual Eur. Sympos. Algorithms (ESA)*, No. 2161. *Lecture Notes in Computer Science*. Springer, Heidelberg, Germany, 182–193.

Lancia, G., R. Carr, B. Walenz, S. Istrail. 2001b. 101 optimal PDB structure alignments: A branch-and-cut algorithm for the maximum contact map overlap problem. *Proc. Fifth Annual Internat. Conf. Comput. Biol.* ACM Press, New York, 193–202.

Lau, K., K. Dill. 1989. A lattice statistical mechanics model of the conformational and sequence spaces of proteins. *Macromolecules* **22** 3986–3997.

Lippert, R., R. Schwartz, G. Lancia, S. Istrail. 2002. Algorithmic strategies for the SNPs haplotype assembly problem. *Briefings Bioinformatics* **3**(1) 23–31.

Molecular Biology Notebook. 2000. *Molecular Biology Notebook*. Rothamsted Research, World Wide Web, http://www.iacr.bbsrc.ac.uk/notebook/courses/guide/.

Neumaier, A. 1997. Molecular modeling of proteins and mathematical prediction of protein structure. *SIAM Rev.* **39** 407–460.

Ngo, J., J. Marks, M. Karplus. 1994. Computational complexity, protein structure prediction, and the Levinthal paradox. K. Merz, Jr., S. L. Grand, eds. *The Protein Folding Problem and Tertiary Structure Prediction*, Ch. 14. Birkhäuser, Boston, MA, 433–506.

Notredame, C., D. Higgins. 1996. SAGA: Sequence alignment by genetic algorithm. *Nucleic Acids Res.* **24**(8) 1515–1524.

Nussinov, R., B. Ma, H. Wolfson. 2002. Computational methods for docking and applications to drug design: Functional epitopes and combinatorial libraries. T. Jiang, Y. Xu, M. Zhang, eds. *Current Topics in Computational Molecular Biology*. MIT Press, Cambridge, MA, 502–524.

Patton, A., W. Punch, E. Goodman. 1995. A standard GA approach to native protein conformation prediction. L. Eshelman, ed. *Proc. Sixth Internat. Conf. Genetic Algorithms*, Morgan Kauffman, San Francisco, CA, 574–581.

Pedersen, C. 1999. *Algorithms in Computational Biology*. Ph.D. thesis. Department of Computer Science, University of Aarhu, Denmark. Available at http://www.brics.dk/~cstorm/.

Pevzner, P. 2000. *Computational Molecular Biology*. MIT Press, Cambridge, MA.

Piccolboni, A., G. Mauri. 1998. Application of evolutionary algorithms to protein folding prediction. *Lecture Notes in Computer Science*, No. 1363, Springer-Verlag, Heidelberg, Germany, 123–136.

Rabow, A. A., H. A. Scheraga. 1996. Improved genetic algorithm for the protein folding problem by use of a Cartesian combination operator. *Protein Sci.* **5** 1800–1815.

Rizzi, R., V. Bafna, S. Istrail, G. Lancia. 2002. Practical algorithms and fixed-parameter tractability for the single individual SNP haplotyping problem. R. Guigo, D. Gusfield, eds. *Proc. 2nd Annual Workshop on Algorithms in Bioinformatics (WABI), Lecture Notes in Computer Science*, No. 2452, Springer-Verlag, Heidelberg, Germany, 29–43.

Sankoff, D., R. Cedergren, Y. Abel. 1990. Genomic divergence through gene rearrangement. *Molecular Evolution: Computer Analysis of Protein and Nucleic Acid Sequences*, Academic Press, New York, 428–438.

Setubal, J., J. Meidanis. 1997. *Introduction to Computational Molecular Biology*. Brooks/Cole, Pacific Grove, CA.

Thompson, J., T. Gibson, D. Higgens. 1995. *Clustal W version 1.5*. World Wide Web, http://www.cmbi.kun.nl/bioinf/CLUSTALW/clustalw-4.html.

Unger, R., J. Moult. 1993a. A genetic algorithm for 3D protein folding simulations. *Proc. Fifth Annual Internat. Conf. Genetic Algorithms*, San Francisco, CA, 581–588.

Unger, R., J. Moult. 1993b. Genetic algorithms for protein folding simulations. *J. Mol. Biol.* **231**(1) 75–81.

Venter, J. et al. 2001. The sequence of the human genome. *Science* **291** 1304–1351.

Wang, L., T. Jiang. 1994. On the complexity of multiple sequence alignment. *J. Comput. Biol.* **1** 337–348.

Wang, L. S., T. Warnow. 2001. Estimating true evolutionary distances between genomes. *Proc. Annual ACM Sympos. Theory of Comput. (STOC)*, ACM Press, New York, 637–646.

Willis, L., ed. 2000. *Biology Hypertextbook*. Massachusetts Institute of Technology, http://esg-www.mit.edu:8001/esgbio/.

Yue, K., K. A. Dill. 1993. Sequence-structure relationships in proteins and copolymers. *Phys. Rev. E* **48**(3) 2267–2278.

Yue, K., K. A. Dill. 1994. Forces of tertiary structural organization in globular proteins. *Proc. National Acad. Sci. (USA)* **92** 146–150.