Multiple Sequence Assembly from Reads Alignable to a Common Reference Genome

Qian Peng and Andrew D. Smith

Abstract—We describe a set of computational problems motivated by certain analysis tasks in genome resequencing. These are assembly problems for which multiple distinct sequences must be assembled, but where the relative positions of reads to be assembled are already known. This information is obtained from a common reference genome and is characteristic of resequencing experiments. The simplest variant of the problem aims at determining a minimum set of superstrings such that each sequenced read matches at least one superstring. We give an algorithm with time complexity $O(N)$, where N is the sum of the lengths of reads, substantially improving on previous algorithms for solving the same problem. We also examine the problem of finding the smallest number of reads to remove such that the remaining reads are consistent with k superstrings. By exploiting a surprising relationship with the minimum cost flow problem, we show that this problem can be solved in polynomial time when nested reads are excluded. If nested reads are permitted, this problem of removing the minimum number of reads becomes NP-hard. We show that permitting mismatches between reads and their nearest superstrings generally renders these problems NP-hard.

Index Terms—Combinatorics, sequence assembly, haplotyping, chain and antichain, superstring.

1 Introduction

Dramatic developments in high-throughput sequencing technology have opened the door to numerous new applications of DNA sequencing. Many of these applications can be described as "resequencing" experiments, where a reference genome has already been sequenced for the organism of interest or another organism that is closely related. When a reference genome is available, sequenced reads can be aligned to the reference genome and there is no need for de novo sequence assembly. Applications that depend on aligning reads to a reference genome include ChIP-seq [39], RNA-seq [36] and individual variation studies [27]; these sequencing applications have had a significant impact on genome sciences in recent years. As long as a reference genome is available, no assembly is required for these applications. In certain other sequencing applications, while traditional sequence assembly is not required, relationships between individual reads must be considered despite the existence of a reference genome. Here we examine algorithmic problems motivated by sequencing applications where the goal is to assemble multiple closely related genomes present in the same sequenced sample. We assume that reads can be mapped to a common reference genome, so the relative positions of reads are known, but work remains to determine which reads correspond to the same superstrings. This type of assembly problem emerges in a surprising variety of biological data analysis contexts.

The simplest such problems are associated with haplotype inference, and are already both well-known and well-studied. Haplotype inference seeks to infer a pair of distinct haplotypes using reads sequenced from a diploid organism. Haplotype inference is of great medical importance, both to help understand the relationship between genotypes and disease phenotypes [18] and as the basis for personalized medicine [4]. Various optimization problems have been formulated for haplotyping a population of individuals from the same species or closely related species [1], [26], [11], [33] and for haplotyping a single individual [31]. While these problems are often NP-hard, some heuristics have been effectively employed in practice. For example, a greedy heuristic with iterative refinement of the initial solution was employed to infer the haplotypes of heterozygous loci of an individual genome [32]. Another method that has proven effective in practice is to use a general technique founded in statistics, such as Markov Chain Monte Carlo [3]. The difficulty of these problems seems to result from sequencing errors, and one general approach to address sequencing errors is to include a stage of either removal of erroneous reads or error correction [38]. For example, the *minimum SNP removal* (single nucleotide polymorphism) and *minimum fragment removal* variants were defined for the individual haplotyping problem as a means of producing data that may be assumed error-free [2].

Research on haplotype inference has focused on diploid species. Polyploidy refers to the state of an organism having more than two sets of homologous chromosomes [37]. Polyploid species have been found among plants, animals and fungi. For example, some species of the sturgeon order *Acipenserformes* have up to 500 chromosomes and are classified as functional octaploids with eight copies of each chromosome [34]. Several degrees of ploidy have been observed among frogs, and *Xenopus ruwenzoriensis* has 12 sets of homologous chromosomes [29]. The phenomenon is

- *Q. Peng is with the Department of Computer Science & Engineering, University of California, San Diego, 9500 Gilman Drive, Mail Code 0404, La Jolla, CA 92093-0114. E-mail: qpeng@cs.ucsd.edu.*
- *A.D. Smith is with the Division of Biological Sciences, University of Southern California, 1050 Childs Way, RRI 201, Los Angeles, CA 90089. E-mail: andrewds@usc.edu.*

Manuscript received 8 Jan. 2010; revised 21 Apr. 2010; accepted 1 Aug. 2010; published online 18 Oct. 2010.
For information on obtaining reprints of this article, please send e-mail to: tcbb@computer.org, and reference IEEECS Log Number TCBB-2010-01-0008. Digital Object Identifier no. 10.1109/TCBB.2010.107.

particularly common in plants, where estimates indicate that more than 70 percent of species of flowering plants have had some ploidy increase in their history [35]. Polyploidy also has important implications for crop improvement due to the associated evolutionary advantages [45]. These advantages are derived from the generation of genomic diversity through gene duplication, but without negatively affecting dosage relationships of functionally related genes [37]. To understand the role of polyploidy in evolution requires methods for inferring haplotypes of polyploid organisms. However, traditional haplotype inference algorithms are specifically restricted to pairs of haplotypes, and cannot be directly applied to polyploid species.

In the field of metagenomics the aim is to understand heterogenous populations of organisms by sequencing samples from those populations [43]. Most commonly the focus is on bacterial populations, for example, those in soils or deep-ocean niches [44] or those living in the human gut [23]. Specific goals of metagenomic experiments include quantifying genomic diversity, identifying species present in a population and quantifying the relative frequencies of species. Because so many bacterial and viral genomes already exist in sequence databases, reads from metagenomics projects can often be mapped to a reference genome. Frequently the reference consists of a specific gene (e.g., 16S rRNA) common to all organisms of interest. Sequenced reads can be mapped to this gene, and matches and mismatches between reads can be used to distinguish the individual species in the sample. For an unknown number of haplotypes in sequencing of virus genomes, Eriksson et al. (2008) reconstructed haplotypes assuming error-free reads (following an error correction stage) with relative positions known prior to the assembly stage [10].

The field of epigenomics presents yet another example of an assembly problem where multiple sequences must be assembled but with the assistance of a reference genome. DNA methylation in mammals occurs primarily at CpG dinucleotides. Healthy differentiated cells have high levels of methylation genomewide, with small regions of very low methylation. These regions of low methylation are usually at CpG islands (which have a high density of CpG dinucleotides) and occur commonly at gene promoters and enhancer regions. Aberrant methylation is a general feature of cancer genomes; a greater understanding of methylation patterns in cancer genomes may lead to both new therapies and new diagnostic markers based on the detection of methylation-based changes occurring early in tumorigenesis [30]. Because DNA methylation is replicated through mitosis, but at a lower fidelity than the DNA itself, DNA methylation patterns can act as markers for individual clones, providing a basis for tracing stem-cell expansion and tumor growth [46], [42], [28]. Making use of methylation patterns in this way requires determining methylation patterns associated with individual cells or cells from the same clone. Bisulfite sequencing can be used to identify methylation states at CpGs covered by individual reads, but to identify methylation patterns of individual cells requires some way of assembling the binary (methylated CpG versus unmethylated CpG) patterns inside individual reads. Since any sample may contain an arbitrary number of distinct methylation patterns, and the reads can be mapped back to a reference genome, this problem represents another example of assembling multiple sequences (in this case sequences of methylation states) from reads mapping to the same reference.

The rest of this paper is organized as follows: In Section 2, we formally define four variations on our central problem. In Section 3, we treat the most basic variant, where the goal is to find the minimum number of superstrings such that all reads are consistent with at least one. We present a novel $O(N)$ time algorithm, where N is the sum of the lengths of reads given as input. This is a linear time algorithm—an important feature since potential inputs may reach one hundred million reads, and continue to grow. In Section 4, we examine the problem of identifying a set of k superstrings such that the greatest number of reads match at least one superstring. Under the assumption that reads are not nested (a concept defined in Section 2), we describe a polynomial time algorithm for this variant. The algorithm is based on the elegant method of Andras Frank developed to unify earlier results about chains and antichains of partially ordered sets [17]. In Section 4.3, we show that when nested reads are permitted, the problem becomes NP-complete.

2 Background and Problem Definitions

DNA-sequencing experiments determine the sequences of nucleotides appearing in fragments of DNA molecules from some biological sample. When one of these DNA fragments is examined by a sequencing instrument, the sequence of nucleotides is produced as a string called a *read*. The details of how the reads are produced from the sample of DNA molecules depend on the exact sequencing technology used. Regardless of the technology, each sequencing experiment produces a set of reads—possibly an extremely large set. The reads provide information to help understand some aspect of the DNA molecules from the original sample of cells. The current "second-generation" sequencing technology has vastly improved both throughput and cost of sequencing experiments [41]. Second-generation technologies typically produce tens to hundreds of million reads per experiment at a cost currently accessible to most labs, and at a cost that is still falling rapidly.

The vast numbers of reads produced by second-generation sequencing technologies currently have lengths ranging roughly between 50 and 500 nucleotides. Based on recent trends, it seems likely that both read lengths and number of reads will continue to increase, with technologies currently under development promising order of magnitude improvements [21], [40].

Many applications of second-generation sequencing depend on the existence of a *reference genome*. Reads are aligned to the reference genome in a process usually called *mapping*. The process of mapping reads identifies the location in the reference genome most similar in sequence to each read; this location is presumed to be the genomic origin of the fragment sequenced to produce the read. Mapping cannot require perfect matching between reads and the reference genome, however, for two reasons. First, the sequencing instruments make errors when identifying nucleotides at each position of a read. Fortunately, the accuracy of sequencing instruments constantly improves. Second, individual organisms, even within the same species, have differences in their genomes. The amount of individual genomic variation differs for different species. As we have noted in the introduction, many applications of second-generation sequencing seek to identify these variations and understand their effects on important phenotypes, such as diseases. Other applications, including assembling a new species using the sequence of a previously assembled related species, attempt to be robust to differences in order to leverage the existing reference genome. In these applications, the differences between reads and the reference genome are critical to the experiments, and, therefore, it is common to have reads with distinct sequences mapped to the same location in the reference genome.

Problems we consider are based on resequencing applications where the sequenced sample is somehow heterogenous. In our introduction we described several contexts where such problems can emerge. We will use the example of inferring haplotypes for polyploid species to give some intuition for the computational problems we define. The different haplotypes in a polyploid will differ from each other, and also differ from any single reference genome, but will have sufficient similarity that they can all be mapped to a single genome. The practical task is to infer the original sequences by partitioning the mapped reads according to the haplotypes from which they originated. Clearly such tasks depend on having sufficient divergence between the original sequences that we may distinguish them.

Throughout this paper, we use the notation $x[i]$ to denote the letter appearing at position i of a string x, and for positions $i \leq j$, we let $x[i, j]$ denote the substring of x from i to j, inclusive.

Let R be a set of strings, which we assume are reads produced in a sequencing experiment. Define $n = |R|$ and for each $r_i \in R$ we use the notation $|r_i|$ for the length of r_i (and similar notation for the length of strings in general). We use the symbol S to denote a set of superstrings (contiguous sequences or contigs) we wish to assemble using the reads of R. We use m to indicate the length of the reference genome, which is an upper bound on the length of members of S.

Recall that the reads have been mapped to some common reference genome. We define the *position function*

$$p : R \mapsto \{(a, b) : 1 \leq a \leq b \leq m\}$$

to indicate the position in the reference genome where each read maps. For each $r_i \in R$, if $p(r_i) = (a_i, b_i)$, then r_i maps to the reference starting at position a_i and ending at $b_i = a_i + |r_i| - 1$. These are closed intervals. For convenience when we use the notation a_i and b_i it will be implicit that for a specific $r_i, p(r_i) = (a_i, b_i)$, and the identity of the specific r_i will be clear from the subscripts.

We define the *distance* between a read $r \in R$ and a superstring $s \in \mathcal{A}^m$, where $\mathcal{A}$ is the sequence alphabet, as the number of corresponding positions where r and s disagree:

$$d_p(r, s) = |\{x : r[x] \neq s[a + x - 1] \text{ where } \\ p(r) = (a, b) \text{ and } 1 \leq x \leq |r|\}|.$$

This is the Hamming distance between r and the corresponding substring of s, and is only defined if s is sufficiently long that $s[a + x - 1]$ exist. Note that insertions and deletions are ignored. We also define the *consistency* relation between r and s as the symmetric Boolean relation

$$C_p(r, s) \Leftrightarrow d_p(r, s) = 0,$$

and when $C_p(r, s)$ we say r and s are consistent with each other. We also extend the notion of consistency to pairs of reads. For any two reads $r, r' \in R$,

$$C_p(r, r') \Leftrightarrow \exists s \in \mathcal{A}^m : C_p(r, s) \wedge C_p(r', s).$$

We now formally define the first and most basic problem variant we will address.

Referenced Multiple Assembly (RMA)
Input: A set R of strings, $|R| = n$, a position function p and a positive integer k.
Objective: Determine whether there exists a set S of strings, with $|S| = k$, having the property that for any $r \in R$ there exists an $s \in S$ such that s is consistent with r.

One optimization version of RMA seeks to minimize k, which in several situations corresponds to a most parsimonious set of superstrings to explain the observed reads. The optimal solution in many cases may not be unique.

By defining p as a function we have required that each read be mapped to a unique location in the reference genome. It is possible that a read aligns equally well to more than one location, and in this case we say the mapping is ambiguous. Depending on the sequencing application, there may be a small or large proportion of reads that map ambiguously. Although certain sequencing applications can make use of ambiguously mapping reads, the most common approaches to deal with them in practice are to either discard them or to select one among multiple optimal mapping locations. We assume that some strategy has been used for dealing with ambiguously mapping reads, and therefore each read has exactly one mapping location. We also remark that as read lengths increase, the proportion of ambiguously mapping reads drops rapidly.

An example instance of RMA is shown in Fig. 1. In this example the reads all have the same length (17 nt) and are all alignable to a single reference. When the reads are restricted to the informative positions where at least one pair of reads differs, the lengths of the reads from Fig. 1a is reduced in Fig. 1b to between 3 nt and 5 nt. This example illustrates how a simple intuitive greedy approach does not guarantee to find a smallest set of superstrings consistent with all reads. This greedy algorithm proceeds as follows: Start by selecting any read having no others preceding it in order of the position function p (sorted on first coordinate, then second). Let that read be the beginning of the first superstring. Then grow that superstring by sequentially appending letters from a read that is both consistent with, and has the greatest overlap with, the superstring. Once no remaining reads can be used to extend the superstring, a new superstring is constructed in the same manner from the remaining reads. A result of applying this greedy algorithm can be seen in Fig. 1c, where 4 superstrings have been assembled. The minimum number in this example is 3, which can be seen in Fig. 1d.

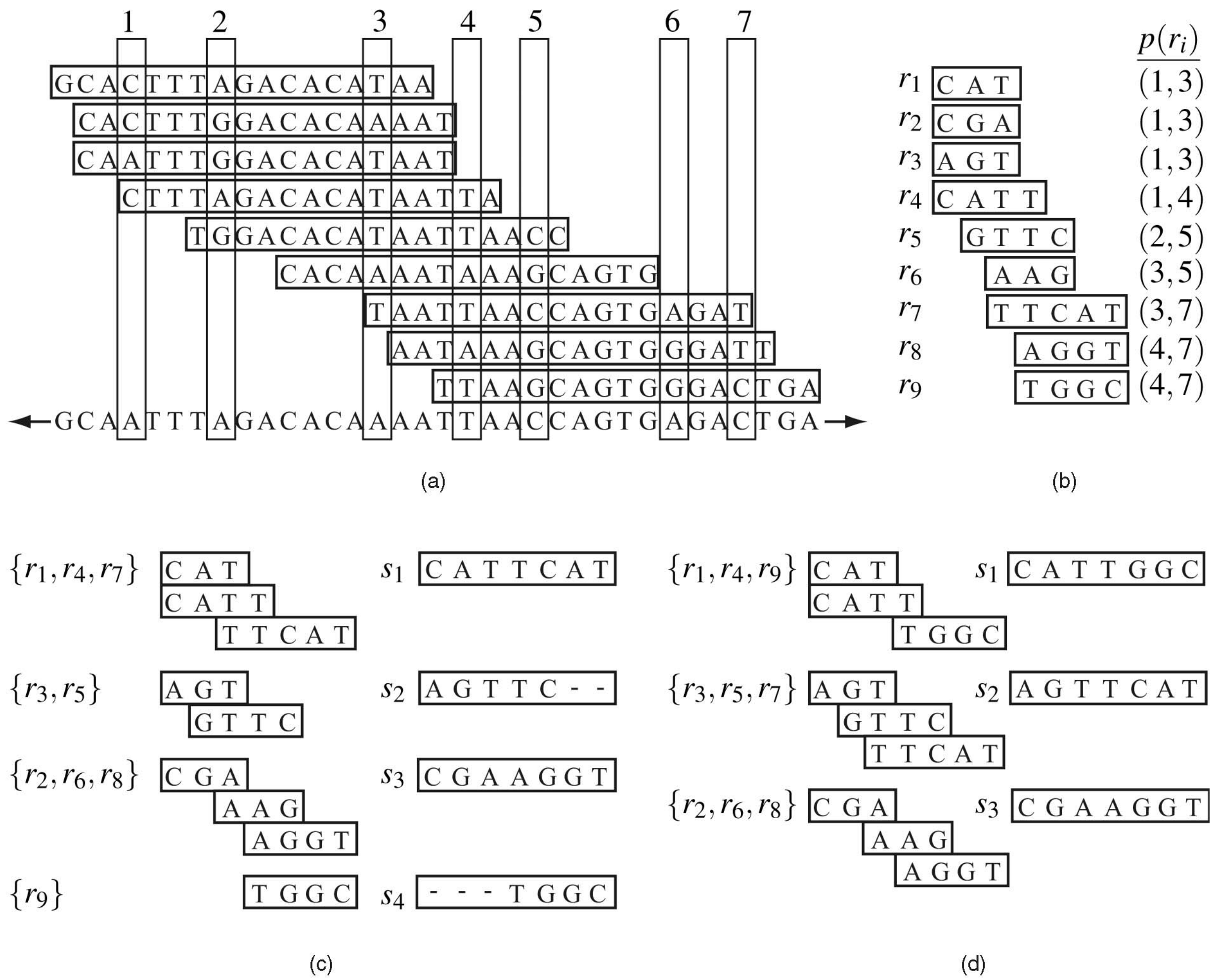


Fig. 1. An example illustrating the referenced multiple assembly problem. This example reflects the practical task of inferring a most parsimonious haplotype set for a polyploid species. (a) A set of reads from a heterogenous sample (e.g., polyploid species) aligned to a reference genome. (b) Reads when restricted to only those positions that differ between reads, along with a position function, relative only to those informative positions. (c) A set of four superstrings assembled from the reads ("-" represents positions unconstrained by the reads). (d) A set of three superstrings (the minimum) assembled from the reads; this represents a most parsimonious set of haplotypes for a polyploid species.

Observe that in Fig. 1c, the superstrings s_2 and s_4 contain "-" symbols, which indicate gaps in the assembled superstrings. Those are positions not covered by any reads that must match that superstring. We assume that these gaps are filled with arbitrary letters. The complexity of an algorithm is bounded from below by the size of its output, and explicitly filling the gaps would mean that any set of k superstrings has size km. Our algorithms fill these gaps in some concise way, such as indicating the positions and size of gaps, rather than explicitly placing letters in them.

The example of Fig. 1 also helps to illustrate another subtle point. When the reads are reduced to informative positions, where the definition of "informative" may depend on the application, then the reads we process may be much shorter than the actual reads produced by the sequencing instrument. In addition, we may observe a pair of reads that differ at most or all of their positions, despite covering the same positions (e.g., r_5 and r_6 in the example). Another possibly counterintuitive observation is that when the reads are reduced to some subset of their positions, then among the reads that must be assembled, two reads at different positions may appear identical at the sequence level. While this can arise as an artifact of eliminated non-informative positions from the reads, in our definition for the RMA problem we make no presumptions on why identical reads may be at different positions.

In what follows, we will use existing results about partially ordered sets (posets), and define these concepts here.

Definition 1 (Partially Ordered Set; Poset). *A partially ordered set, or poset, is a pair $(X, \leq)$, where X is a set and $\leq$ is a relation satisfying:*

1. $x \leq x$ *(reflexivity)*
2. *if* $x \leq y$ *and* $y \leq x$ *then* $x = y$ *(antisymmetry)*
3. *if* $x \leq y$ *and* $y \leq z$ *then* $x \leq z$ *(transitivity)*

for all $x,\ y,\ z \in X$. *If* $(X, \leq)$ *is a partially ordered set, a subset* X' *of* X *is called a chain if* $x \leq y$ *or* $y \leq x$ *for all* $x, y \in X'$. *A subset* X' *of* X *is called an antichain if for all* $x, y \in X'$, *neither* $x \leq y$ *nor* $y \leq x$.

We define the *position order* relation $\leq_p$ as follows: For reads $r_i, r_j \in R$, with $p(r_i) = (a_i, b_i)$ and $p(r_j) = (a_j, b_j)$,

$$r_i \leq_p r_j \Leftrightarrow (a_i < a_j \vee (a_i = a_j \wedge b_i \leq b_j)).$$

We also define an augmentation of the position order to account for the sequence of each read:

$$r_i \leq_q r_j \Leftrightarrow (a_i < a_j \vee (a_i = a_j \wedge b_i < b_j) \vee (a_i = a_j \wedge b_i = b_j \wedge r_i \leq_{\text{lex}} r_j)), \quad (1)$$

where $\leq_{\text{lex}}$ is the lexicographic order for strings. When we must assume the reads are presented to an algorithm in some particular format, we assume pairs of the form $(r, p(r))$, and that the reads are presented in $\leq_q$ order. This is the most natural way to store reads after they have been mapped to a reference genome: with sequence and mapping location together, sorted according to the mapping location. Note that the lexicographic order is not required in our algorithms for strings that are mapped to the same location and have same length.

Let $r \in R$, with $p(r) = (a, b)$. If $a \leq x \leq b$, we say that r *covers* position x. We say that position x in the reference genome and position x' in r are *corresponding* positions, if $x = a + x' - 1$. This can be visualized as position x' in read r positioned directly above position x in the reference genome when r is aligned to the genome. For reads $r, r' \in R$, if position x in r corresponds to genomic position y, and genomic position y corresponds to position x' in r', then position x in r corresponds to position x' in r'.

Under certain assumptions, we can define a useful partial order on the reads R. For all $r_i, r_j \in R$ define the relation

$$r_i \preceq r_j \Leftrightarrow (r_i \leq_p r_2 \text{ and } C_p(r_1, r_2)).$$

When all reads have the same length then $(R, \preceq)$ is a poset. The assumption of uniform length reads is often reasonable, as some popular sequencing technologies currently produce reads of uniform length. Two of the most popular current technologies (the Illumina/Solexa technology and the ABI SOLiD technology) each produces reads of equal lengths, although we remark that postprocessing in both technologies is a quality clipping which may result in uneven read length when low-quality ends of reads are removed. Technologies that sequence by synthesis [21] will often produce the majority of reads with the same length.

The benefit of the poset $(R, \preceq)$ is that it allows the RMA problem to be transformed into a well-studied combinatorial problem. The following famous result about posets provides a convenient characterization for the RMA problem.

Theorem 1 (Dilworth's Theorem [8]). *Suppose P is a partially ordered set, and no antichain of P contains more than k elements. Then P can be partitioned into k disjoint chains.*

The maximum size of an antichain of a poset is called its width. Consequently, if the width of poset $(R, \preceq)$ is at most k, then there exists a set S of superstrings, with $|S| = k$, such that each $r \in R$ is consistent with at least one $s \in S$. A minimum chain partition of $(R, \preceq)$ can be found using the device of *maximal bipartite matching*, as shown in the proof of Dilworth's Theorem given by Fulkerson [20]. This method has been employed by Eriksson et al. (2008) in the context of viral metagenomics to reconstruct a set of haplotypes that best explain the error-corrected reads [10]. The approach taken was to explicitly convert the RMA problem into a directed acyclic graph (a natural poset representation) and obtain a minimum chain partition of the poset using bipartite matching. This method has cubic time complexity in the number of reads given as input. For a given value of k, Felsner et al. describe an algorithm that can find a partition of a poset into k chains, or determine that none exists, in $O(kn^2)$ time [12]. The method of Felsner achieves the fastest asymptotic time performance as a function of n.

Although having reads with equal length is sufficient for $(R, \preceq)$ to be a poset, this condition is not necessary. A more general condition that is also sufficient is that reads cannot be *nested*. We say that read r is nested in read r' if

$$a' < a, \ b < b' \quad \text{and} \quad C_p(r, r'),$$

where $p(r) = (a, b)$ and $p(r') = (a', b')$. When the RMA problem is used to model the inference of multiple haplotypes, the set R may actually correspond to subsequences of reads, restricted to positions of informative SNPs. In this case, members of R may effectively have different lengths, but no nesting will be observed as long as the actual reads have equal lengths. It is straightforward to show that when no nesting is permitted, the order $(R, \preceq)$ is a partial order on the reads, regardless of possible variation in lengths of reads.

Proposition 1. *If nesting of reads is not permitted, $(R, \preceq)$ is a poset.*

Proof. Reflexivity and antisymmetry of $\preceq$ over R can be demonstrated regardless of whether nesting is permitted. To verify transitivity, let $r_i, r_j, r_k \in R$ and assume $r_i \preceq r_j$ and $r_j \preceq r_k$. It is immediate from the definition of $\leq_p$ that $a_i \leq a_j \leq a_k$. Because nesting of reads is not permitted, $b_i \leq b_j \leq b_k$, so $r_i \leq_p r_k$. If the interval $p(r_j)$ intersects at most one of $p(r_i)$ and $p(r_k)$, then $p(r_i)$ and $p(r_k)$ do not intersect and $C_p(r_i, r_k)$ holds trivially. Otherwise we assume $p(r_i)$ and $p(r_k)$ have a nonempty intersection with $p(r_j)$. Since both $C_p(r_i, r_j)$ and $C_p(r_j, r_k)$,

$$p(r_i) \cap p(r_k) \subseteq p(r_i) \cap p(r_j) \cap p(r_k)$$

is a sufficient condition for $C_p(r_i, r_k)$, which holds trivially if $p(r_i) \cap p(r_k)$ is empty. If not, once more using the premise of nonnested reads,

$$\begin{aligned} p(r_i) \cap p(r_j) \cap p(r_k) &= (a_j, b_i) \cap (a_k, b_j) \\ &= (a_k, b_i) \\ &= p(r_i) \cap p(r_k) \end{aligned}$$

and the sufficient condition is seen to hold. □

A sequencing error occurs when the sequencing instrument incorrectly identifies the nucleotide appearing at a particular position in a read. Although different sequencing technologies use different methods to determine the nucleotides of a read, every technology has some error rate. The RMA problem can be reformulated to explicitly account for sequencing errors. In the following variant, the goal is to minimize the total number of mismatches occurring between reads and the superstring they match most closely.

Minimum Distance RMA (MD-RMA)

Input: A set R of strings, $|R| = n$, a position function p and a positive integer k.

Objective: Find a set S of strings, with $|S| = k$, minimizing

$$\sum_{r \in R} \min_{s \in S} d_p(r, s).$$

While this problem seeks to minimize a sum of distances (i.e., mismatches), a less natural objective could be to bound the allowed sum of distances and seek to minimize the value of k. MD-RMA is actually a generalization of the parameterized binary minimum error correction (PBMEC). The difference between MD-RMA and PBMEC is that PBMEC assumes a binary alphabet. Unfortunately, it has already been shown that PBMEC is NP-hard [5], and these results extend directly to MD-RMA.

An alternative way to incorporate mismatches between reads and superstrings has an objective function that minimizes the maximum number of mismatches between any read and the closest superstring.

Bounded Distance RMA (BD-RMA)
Input: A set R of strings, $|R| = n$, a position function p and a positive integer k.
Objective: Find a set S of strings, with $|S| = k$, minimizing the value D such that for all $r \in R$,

$$\min_{s \in S} d_p(r, s) \leq D.$$

As with the MD-RMA problem, we could include the value of D as part of the problem instance, and the objective to minimize k. Bounding the value of D may be a natural restriction (more so than bounding the sum in MD-RMA) because frequently in practice reads are used if they have at most some number of mismatches with respect to the reference, reflecting the notion that a few sequencing errors in reads can be permitted, but too many might be indicating a systematic problem.

We show BD-RMA is NP-hard through a trivial reduction from a problem known as the minimum radius of a code. Let $\mathcal{C}$ be a binary code of length m, that is, $\mathcal{C} \subseteq \{0,1\}^m$. For two vectors $u, v \in \{0,1\}^m$, denote the Hamming distance between them by $d(u,v)$. Define the Hamming ball of radius D and center $u \in \{0,1\}^m$ as the set

$$B(u, D) = \{v \in \{0,1\}^m : d(u, v) \leq D\}.$$

Radius of code $\mathcal{C}$ is the smallest integer D that $\mathcal{C} \subseteq B(u, D)$ for some vector u. The following problem concerning the radius of a code has been shown NP-complete [16].

Minimum Radius of a Code (MR)
Input: A code $\mathcal{C} \subseteq \{0,1\}^m$ and a positive integer D.
Objective: Determine whether the radius of $\mathcal{C} \leq D$?

When the BD-RMA problem is restricted such that $|r_i| = m$ for all i, and the alphabet is $\{0,1\}$, it is equivalent to the MR problem, which immediately implies that BD-RMA is NP-hard. This leaves open the complexity of the case where each $|r_i|$ is $o(m)$. Because MR places such strong restrictions on BD-RMA, the BD-RMA is likely much harder and would benefit from a deeper complexity analysis.

We do not address the issue of approximability for these NP-hard problems, but make the following remarks: As with many optimization problems involving distances, we can formulate corresponding problems by changing the distance measures in MD-RMA and BD-RMA to similarity measures. The objective then would be to maximize the similarity—the sum of similarities for MD-RMA and the minimum similarity for BD-RMA. The most obvious way to obtain a similarity measure from the Hamming/mismatch distance is to replace $d_p(r, s)$ above with $|r| - d_p(r, s)$. When all reads have the same length, the similarity and distance versions of the problems have coinciding optima. However, when the read lengths vary, the sets of superstrings that optimize the distance may not be those that optimize the similarity.

Another way to account for errors or "noisy" data is to regard some subset of the reads as "bad" in some way. These bad reads may be contamination in the sequenced sample (a problem independent of any technology). Problematic reads may also reflect a distribution of sequencing errors that concentrates a high proportion of errors in a small proportion of the reads—which is plausible because sequencing errors are not independent from each other for certain sequencing technologies.

With these problematic reads in mind, we may desire to identify a set of superstrings that is consistent with all but some relatively small set of reads that we then discard. A most parsimonious solution would then be a set of superstrings that minimizes the number of reads we must discard. The variation of the individual haplotyping problem, known as *minimum fragment removal*, is based on the same idea [2], and we define an analogous variation on RMA.

Minimum Fragment Removal RMA (MFR-RMA)
Input: A set R of strings, $|R| = n$, a position function p and a positive integer k.
Objective: Find a set S of strings, with $|S| = k$, maximizing the size of $R' \subseteq R$ such that for all $r \in R'$, there exists an $s \in S$ with $C_p(r, s)$.

The difference between MFR-RMA and the MFR objective for the individual haplotype assembly problem is that the latter fixes the value $k = 2$, assumes a binary alphabet and may allow "gaps" in the reads, where no letter appears (similar to allowing wildcard letters that can match any letter of the alphabet). When gaps are not permitted, the MFR-RMA problem is a generalization of the MFR problem in haplotyping. Bafna et al. (2005) gave an $O(n^2m + n^3)$ time algorithm for the gapless case, where n is the number of fragments (analogous to the reads in our terminology) and m is the length of the haplotype (analogous to the superstrings we assemble) [2]. We address the MFR-RMA problem in Section 4.1, where we show that MFR-RMA can be solved in polynomial time when reads cannot be nested. In Section 4.3, we also show that allowing nested reads immediately renders it NP-complete.

3 The Referenced Multiple Assembly Problem

In this section, we describe a linear time algorithm for the RMA problem. By linear, we mean a linear function of the number of letters in the input. While we assume that the alphabet is of constant size and that any numbers involved (e.g., values of the position function p) can be manipulated in constant time with operations of the unit-cost RAM model [6], our algorithm does not require reads to be of equal length.

The approach we take here is to construct the superstrings simultaneously. Reads are processed in $\leq_q$ order, and for each read, some superstring may be extended by adding letters so that the read is consistent with the superstring. We, therefore, ensure that a read is consistent with at least one superstring before considering the next read. Recall that we demonstrated in Section 2 that a particular greedy strategy does not result in the smallest number of superstrings. We show first that our strategy of simultaneously growing multiple superstrings results in an optimal solution, then we describe an algorithm to accomplish this strategy in linear time.

3.1 The Strategy of Conservative Extensions

We maintain a set of superstring prefixes as a partial solution that is modified as each read is processed. Initially, the partial solution is empty. After read r_i is processed, the current partial solution is the set S_i and at least one member of S_i matches each $r_j \leq_q r_i$. At each iteration, S_i is obtained by modifying S_{i-1}.

Two kinds of modifications are used—extending an existing superstring from S_{i-1} or adding a new superstring. Suppose, there exists an $s \in S_{i-1}$ such that letters can be appended to s so that the resulting string s' is consistent with r_i. Then we let $S_i = (S_{i-1} \setminus \{s\}) \cup \{s'\}$. In this case, r_i is a suffix of s', and we say that r_i has extended s. Note that the s we have specified may not be unique, and we will explain shortly how to select an appropriate s.

If $p(r_i) = (a_i, b_i)$, and $|s| < a_i$, then simply adding letters from r_i to s at corresponding positions will result in a gap in s'. We assume there is some concise encoding used to fill such a potential gap, otherwise the length m of superstrings would impact time complexity.

If no member of S_{i-1} can be made to match r_i by appending letters, then each member of S_{i-1} already mismatches r_i at some corresponding position. In this case, we let s be the empty string, and append letters to s resulting in s' being consistent with r_i (again using some encoding to fill in gaps not specified by the requirements of consistency with r_i). In this case, $|S_i| = |S_{i-1}| + 1$.

When read r_i is processed, the positions in the interval (a_i, b_i) completely determine whether zero, one, or multiple members of S_{i-1} can be made consistent with r_i by appending letters. In case multiple members of S_{i-1} can be made consistent with r_i, the procedure we specified above is ambiguous. The rule we use is to always select a *longest* member of S_{i-1} that can be extended to match r_i. We refer to this as a *conservative* extension, because using this rule adds the fewest letters to a member of S_{i-1}. It is possible that r_i will already match some member of S_{i-1}, and so a conservative extension would actually append an empty string. An illustration of conservative extensions can be found in Fig. 2.

We now prove this strategy of processing each $r \in R$ in $\leq_q$ order, using conservative extensions, arrives at an S_n of minimal cardinality. We require another definition first. For any set S_i of superstring prefixes, with $p(r_i) = (a_i, b_i)$, a *completion* for S_i is a set Z of strings having length $m - a_i + 1$ with $|Z| \geq |S_i|$ and satisfying for all $s \in S_i$,

$$\exists z \in Z : s[a_i, |s|] = z[1, |s| - a_i + 1],$$

and for all $r_j \in R$, if $r_i \leq_q r_j$,

$$\exists z \in Z : r_j = z[a_j - a_i + 1, b_j - a_i + 1].$$

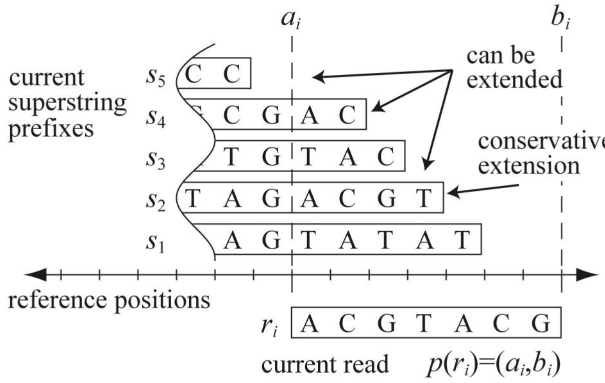


Fig. 2. Example of a *conservative* extension. When r_i is processed $S_{i-1} = \{s_1, s_2, s_3, s_4, s_5\}$. Although s_2, s_4, and s_5 can be extended to match r_i, only s_2 would be a conservative extension, since the number of additional letters that must be appended to s_2 is less than required for s_4 and s_5.

That is, a completion appends letters to current superstring prefixes, and possibly adds new superstrings, so that all remaining reads are consistent with at least one. Notice that a completion may be arbitrarily large, but must contain at least one string for every string in S_i. A *minimal completion* is a completion of minimal cardinality. We make the following observation:

Proposition 2. *The set of completions for S_i is fully determined by $R_{i+1} = R \setminus \{r_1, \ldots, r_i\}$ and the set of suffixes $s[a_i, |s|]$ for each $s \in S_i$. In particular, given R_{i+1} and S_i with completion Z, if we transform S_i into S'_i by removing terminal letters from some $s \in S_i$, then Z is a completion for S'_i.*

Lemma 2. *Suppose a set of superstring prefixes S_{i-1} has a completion of size k. If S_i is obtained from S_{i-1} using the conservative extension rule, then S_i will have a completion of size k.*

Proof. If no member of S_{i-1} can be extended to match r_i, then adding a new member in S_i is necessary, and will not increase the size of the minimal completion.

Suppose, some member of $s_o \in S_{i-1}$ can be extended to match r_i, resulting in S_i having a completion of size k. Let $s_c \in S_{i-1}$ be selected by the conservative extension rule, and suppose $s_c \neq s_o$. Let s'_o and s'_c result from extending s_o and s_c, respectively, to match r_i. Let $S'_{i-1} = S_{i-1} \setminus \{s_o, s_c\}$ and define

$$S_{i(o)} = S'_{i-1} \cup \{s_c, s'_o\} \quad \text{and} \quad S_{i(c)} = S'_{i-1} \cup \{s'_c, s_o\}.$$

Because s'_c and s'_o are identical within the interval $p(r_i)$, any difference in the set of completions for $S_{i(o)}$ and $S_{i(c)}$ is completely determined by the suffixes of s_o and s_c within the interval $p(r_i)$. But the suffix of s_o in the interval $p(r_i)$ is a prefix of the suffix of s_c in the same interval. Therefore, by Proposition 2, the completion set for $S_{i(o)}$ is a subset of the completion set $S_{i(c)}$. In particular, any minimal completion for $S_{i(o)}$ is also a completion for $S_{i(c)}$, so s_c has a completion of size k. □

We point out that any algorithm using the strategy of conservative extensions will ensure, for each read $r_i \in R$, that at least one superstring is consistent with r_i after it has been processed. Therefore, when the algorithm terminates, the set of superstrings constructed will be a valid solution.

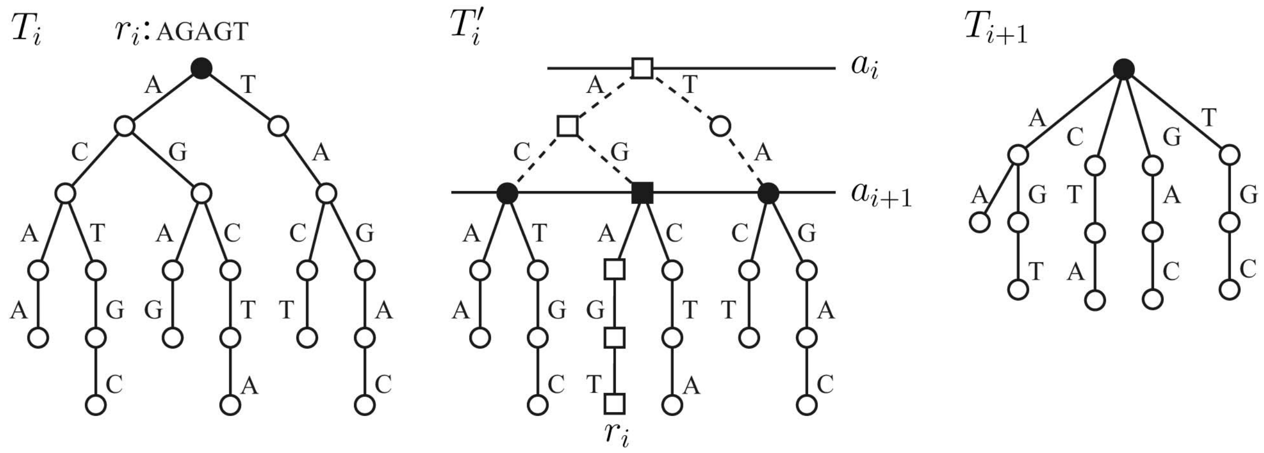


Fig. 3. Example of updating T_i to obtain T_{i+1}. Black nodes are roots of tries. In T'_i, square nodes indicate the path corresponding to r_i : AGAGT. Nodes at a depth less than $a_{i+1} - a_i$ are removed from T_i, leaving the forest of tries T'_i rooted at nodes of depth $a_{i+1} - a_i$. Then, the union of T'_i is taken, resulting in T_{i+1}. This can be done in amortized linear time over all $r_i \in R$.

3.2 Conservative Extensions in Linear Time

Now we describe an algorithm that processes each read in $\leq_q$ order to build each partial solution S_i using conservative extensions. We make use of *trie* data structures [19]. Corresponding to each $r_i \in R$, define T_i as the smallest trie encoding the strings

$$\{r_j[a_i - a_j + 1, |r_j|] : r_j \leq_q r_i \text{ with } r_j \neq r_i,\\ p(r_i) = (a_i, b_i) \text{ and } p(r_j) = (a_j, b_j)\\ \text{and } a_i - a_j + 1 \leq |r_j|\}.$$

Notice that T_i encodes suffixes of other reads overlapping r_i but does not necessarily encode r_i. As T_i is used, it will be updated to encode r_i. Each node of $v \in T_i$ at depth x is labeled with the identity of some $r_j \leq_q r_i$ such that the path from the root to v spells

$$r_j[a_i - a_j + 1, a_i - a_j + x].$$

Each leaf in T_i, therefore, corresponds to a suffix of some $r_j \leq_q r_i$, and since these suffixes also correspond to distinct superstrings in S_{i-1}, we make the correspondence from the leaves of T_i to distinct superstrings in S_{i-1}.

Any superstring in S_{i-1} with no corresponding leaf in T_i is called *free*, and we denote by F the set of free superstrings. We use the notation $S(r_i) = s$ if s was extended to match r_i when r_i was processed. To identify an s that can be extended to match r_i, walk down T_i following the path corresponding to r_i. There are three cases to consider:

1. *All letters of* r_i *are matched.* In this case, r_i is equal to a substring of some $r_j \leq_q r_i$, and we can therefore ignore r_i as it already matches $S(r_j)$.
2. *A leaf* v *is reached without matching all of* r_i. Let r_j label v. Then $S(r_j)$ can be extended to match r_i. Add nodes below v so T_i encodes r_i, extend $S(r_j)$ to match r_i and set $S(r_i) = S(r_j)$.
3. *Matching* r_i *fails at an internal node* v. If $F \neq \emptyset$, extend some $s \in F$ to match r_i and set $S(r_i) = s$. If $F = \emptyset$ create a new $s \in S$ matching r_i and set $S(r_i) = s$. Add nodes below v so T_i encodes r_i.

This algorithm implements conservative extensions, since in step 2 the superstring extended has maximal overlap with r_i. Notice that the work required in each step is $O(|r_i|)$ and following execution of these steps r_i is encoded in T_i. Before processing r_{i+1}, the trie T_{i+1} is obtained from T_i in the following two steps:

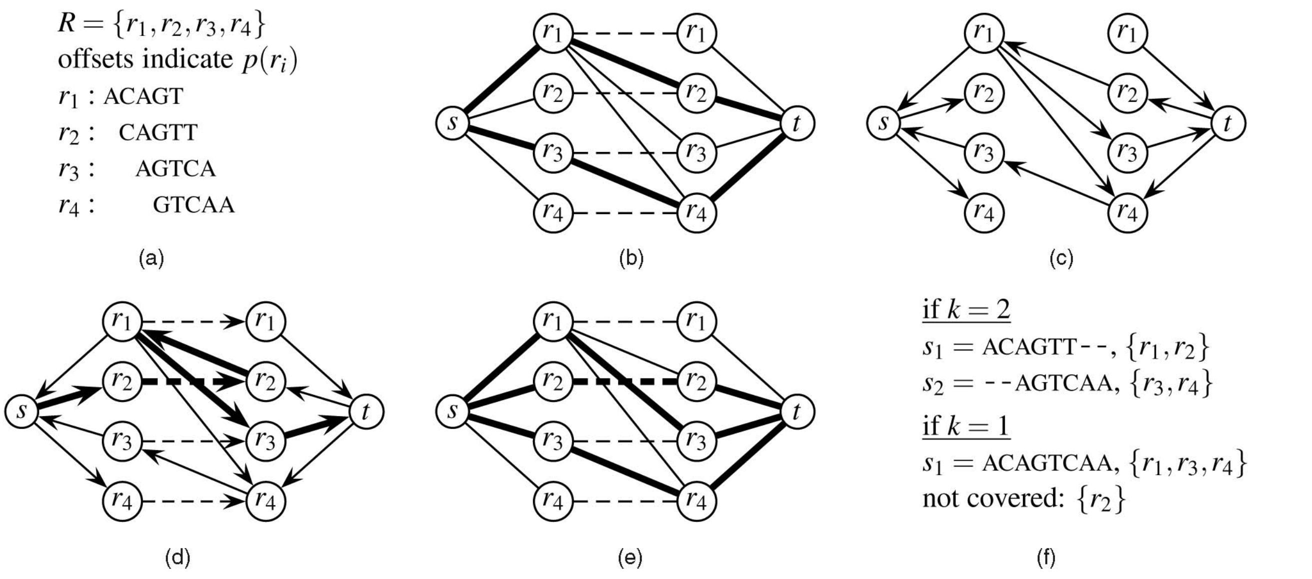
1. T'_i is obtained as the forest of subtries rooted at depth $a_{i+1} - a_i$ in T_i.
2. T_{i+1} is obtained as $\cup T'_i$ by sequentially taking $|T'_i| - 1$ unions of pairs of tries from T'_i.

These steps are partially illustrated in Fig. 3. When taking unions of two tries, we specify one as *persistent* and the other *transient*. The result of the union operation will be the persistent trie with subtries added from the transient trie. Taking the union of tries is done by walking paths common to both tries. When a node is encountered that exists only in the transient trie, the subtrie rooted at that node is added to the persistent trie in constant time (e.g., by updating a link) at the corresponding location.

It is straightforward to see that this method correctly obtains T_{i+1} from T_i, since the forest T'_i contains all suffixes required in T_i and the union of T'_i encodes an identical set of suffixes. What remains is to bound the time required for taking unions of tries.

Lemma 3. *The time required for all trie unions is* $O(N)$.

Proof. We bound the time complexity by bounding node visits. Each node on a path, common to both the persistent and transient tries, is visited once while walking the common path. Nodes on the common path in a transient trie are deleted following the union, and, therefore, will never be visited again. The total time required for walking the common paths in union operations is then bounded by the number of nodes that can be deleted, which cannot exceed the $O(N)$ total nodes created. Each time a subtrie is moved from the transient trie to the persistent trie requires constant time. For each node moved from a transient trie to the persistent trie, the parent node is deleted, and since the $O(N)$ parent nodes that exist through the entire course of the algorithm have $O(N)$ total children (as we assume alphabet size is constant), the total number of such subtrie relocations is bounded by $O(N)$. □



$R = \{r_1, r_2, r_3, r_4\}$
offsets indicate $p(r_i)$
r_1 : ACAGT
r_2 :   CAGTT
r_3 :   AGTCA
r_4 :   GTCAA

if $k = 2$
$s_1 =$ ACAGTT- -, $\{r_1, r_2\}$
$s_2 =$ - -AGTCAA, $\{r_3, r_4\}$
if $k = 1$
$s_1 =$ ACAGTCAA, $\{r_1, r_3, r_4\}$
not covered: $\{r_2\}$

Fig. 4. The method of Frank applied to the MFR-RMA problem when nested reads are excluded. Background on network flows can be found in [7] and [15]. All edges have capacity 1, and when arrowheads are omitted, are directed to the right. Solid edges have cost 0 and dashed edges have cost 1. (a) The set R of reads used in the examples. (b) The flow graph corresponding to R, with a flow value of 2 (thick edges) which is maximum for the permitted cost of 0. (c) With a permitted cost of 0, the residual graph has no augmenting path. (d) An augmenting path exists (bold edges) in the residual graph after increasing permitted cost to 1. (e) The augmented flow (bold edges) has value 3 and cost 1, incurred on (dashed) edge (r_2, r_2) indicating r_2 matches no superstring. (f) The flow of cost 0 corresponds to a pair of superstrings, at least one of which matches each read. Dashes "-" indicate positions not constrained by the reads matching the given superstring. The flow of cost 1 corresponds to a single superstring matching three of the reads.

The conservative extensions strategy was shown correct in Lemma 2 and Lemma 3 showed that this strategy can be accomplished in $O(N)$ time, giving the desired time bound.

Theorem 4. *The referenced multiple assembly problem can be solved in $O(N)$ time, where N is the sum of the lengths of reads.*

4 Removing the Smallest Number of Reads

The objective of MFR-RMA is to determine the minimum number of reads that must be removed from R so that all remaining members are consistent with some size k set S of superstrings. Even when the reads have uniform length, this problem cannot be solved using an approach like the conservative extensions described for solving RMA in Section 3. Consider the MFR-RMA instance depicted in Fig. 4a. When $k = 1$, the optimal solution removes r_2 and is consistent with $\{r_1, r_3, r_4\}$. Without examining r_3 and r_4, however, there is no information to prevent the inclusion of $\{r_1, r_2\}$, which subsequently precludes an optimal solution.

4.1 The Method of Andras Frank

A solution for the MFR-RMA problem can be obtained using the elegant method described by Andras Frank in 1980 for identifying chain and antichain families in partially ordered sets [17]. In particular, Frank's method identifies sets of k chains (or antichains) whose union contains the most elements from the poset. Recall from Section 2 that in the absence of nested reads $(R, \preceq)$ forms a poset, and that a chain in $(R, \preceq)$ consists of a subset of reads, all of which are consistent with a single superstring. In 1976, as two distinct generalizations of Dilworth's Theorem, Greene and Kleitman gave formulas for the maximum cardinality of the union of h antichains and of k chains in a poset [24], [25]. The insight of Andras Frank unified these results and provided an algorithm for simultaneously finding optimal sets of k chains and h antichains [17]. At the center of the algorithm is a generalization of Fulkerson's proof of Dilworth's Theorem [20]. The generalization indicates how finding a maximum weight set of k chains can be reduced to that of finding a *minimum cost flow* in a particular network. By leveraging this reduction, the MFR-RMA problem can be solved in polynomial time when reads are not nested.

Before explaining Frank's reduction, we require additional notation. Let $G = (V, E)$ be a directed graph with $s, t \in V$, and let $\mathrm{in(u)}$ and $\mathrm{out(u)}$ denote the set of edges into and out of node u, respectively. A function $f : E \mapsto \mathbb{R}$ is called an *$s - t$ flow* if $f(e) \geq 0$ for all $e \in E$,

$$\sum_{e \in \mathrm{in}(u)} f(e) = \sum_{e \in \mathrm{out}(u)} f(e) \text{ for all } u \in V \setminus \{s, t\},$$

and the flow *value* is defined as

$$\mathrm{value}(f) = \sum_{e \in \mathrm{out}(s)} f(e).$$

A *capacity function* has the form $c : E \mapsto \mathbb{R}$ and we say a flow f respects capacities c if $f(e) \leq c(e)$ for all $e \in E$. A *cost function* is of the form $\phi : E \mapsto \mathbb{R}$ and the cost of any flow f is defined as

$$\mathrm{cost}(f) = \sum_{e \in E} f(e)\phi(e).$$

The minimum cost flow problem is a fundamental problem in combinatorial optimization, and has received much attention over the past 50 years due to its importance as an abstraction of computational problems arising in transportation and shipping.

Minimum-Cost Flow
Input: A directed graph $G = (V, E)$, a pair of designated vertices $s, t \in V$, a capacity function c, a cost function ϕ, and a target flow value v.
Objective: Find an $s - t$ flow in G respecting capacities c, with $\mathrm{value(f)} \geq \mathrm{v}$ and minimizing $\mathrm{cost(f)}$.

Finding the k chains with largest union is reduced to minimum-cost flow as follows: Associate the network $G = (V, E)$ with poset $(R, \preceq)$, where $V = X \cup Y \cup \{s, t\}$. The sets $X = \{x_1, \dots, x_n\}$ and $Y = \{y_1, \dots, y_n\}$ correspond to reads in R. Edges in the network are defined as

$$E = \{(s, x_i) : 1 \leq i \leq n\} \cup \{(y_i, t) : 1 \leq i \leq n\} \cup \{(x_i, y_j) : r_i \preceq r_j\},$$

and all edges have unit capacity. The costs on each edge of the form (x_i, y_i) is set to 1 and all other edge costs are 0.

Examples of graphs constructed in this way from a set of reads are presented in Fig. 4. Details of network flows and associated algorithms can be found in several texts (e.g., [7], [15]), but we briefly sketch the approach of Ford and Fulkerson [13] for finding a maximum flow. Begin with a (suboptimal) flow and repeatedly find "augmenting paths" whose inclusion increases the value of the flow. An augmenting path can be found by first constructing a "residual graph" where edges with flow already at capacity are reversed, and edges with flow below capacity have their current flow value subtracted from their capacity. An augmenting path is then a path from s to t in the residual graph. Once an augmenting path has been found, it can be added to the flow by increasing or reducing the flow along each edge in the path. When no augmenting path can be found, a flow of maximum value has been found. For rational capacities, this method is easily seen to terminate; if the augmenting path used at each iteration is selected carefully, the number of iterations is polynomial.

Note that since all edges in E have unit capacity and costs in $\{0, 1\}$, all possible flow values and costs must be in $0, \dots, n$. This is a direct corollary of the max-flow min-cut theorem: The maximum flow value is exactly equal to the minimum capacity of a cut separating nodes s and t, where the capacity of a cut is the sum of the capacities of its edges [14].

Correctness of this reduction can be understood through certain relationships that are further detailed elsewhere [17]. Suppose for a flow of value v through G the minimum cost is z. Such a flow is on z edges of type (x_i, y_i) and $v - z$ edges of type (x_i, y_j), with $i \neq j$. In any maximum flow through G, regardless of cost, edges of type (x_i, y_j) form independent sets. As shown in the proof of Dilworth's Theorem given by Ford and Fulkerson [15] (p.62, Lemma 8.1), these independent sets define a set of chains. If (x_i, y_j) is involved in the flow, then r_i and r_j are part of the same chain. For any i, if neither x_i nor y_i are involved in the flow, then r_i is included in the chain decomposition as a singleton set. These relationships imply that some set of $n - v$ chains in the poset covers $n - z$ elements. The set $R' \subseteq R$ that solves the MFR-RMA instance is obtained by simply selecting the reads corresponding to covered elements in the poset, and a set of superstrings can be formed by laying out the reads contained in each of the chains.

To solve the MFR-RMA problem for k superstrings, explicitly construct the poset $(R, \preceq)$ defined in Section 2, which has $O(n^2)$ pairs of comparable elements and can be constructed naively in $O(nN)$ time. Then construct the graph G as described above, which takes $O(n^2)$ time. The most straight-forward way to solve the minimum cost flow problem is by iteratively increasing values of a permitted cost z and finding a maximum flow value for each z (e.g., using the augmenting path method sketched above). The first value of z, for which the optimal flow value is at least $n - k$, gives the maximum number of reads that are consistent with some set of k superstrings.

The original minimum cost flow algorithm of Ford and Fulkerson has time complexity $O(n^4)$ [14]. For general minimum cost flow problems, there have been many algorithmic improvements (e.g., [9], [22]). Since for all practical purposes, $N \ll n^2$, the overall time complexity is dominated by the minimum cost flow algorithm.

Theorem 5. *The minimum fragment removal RMA problem can be solved in $O(nN + f(n))$ time when reads are not nested, where n is the number of reads, N is the sum of the lengths of reads, and $f(n)$ is the time complexity for the minimum cost flow algorithm.*

4.2 Minimum Fragment Removal is NP-Hard when Allowing Nested Reads

Although the MFR-RMA problem is polynomial-time solvable when restricted to reads of uniform length, we show here that allowing variable length reads, which, in general, includes nested reads, renders the problem NP-complete. Our reduction is from the well-known 3SAT problem.

3-SATISFIABILITY (3SAT)
Input: Collection $C = \{c_1, \dots, c_m\}$ of clauses on a finite set U of variables such that $|c_i| = 3$ for all i.
Objective: Determine whether a truth assignment exists for U that satisfies all the clauses in C.

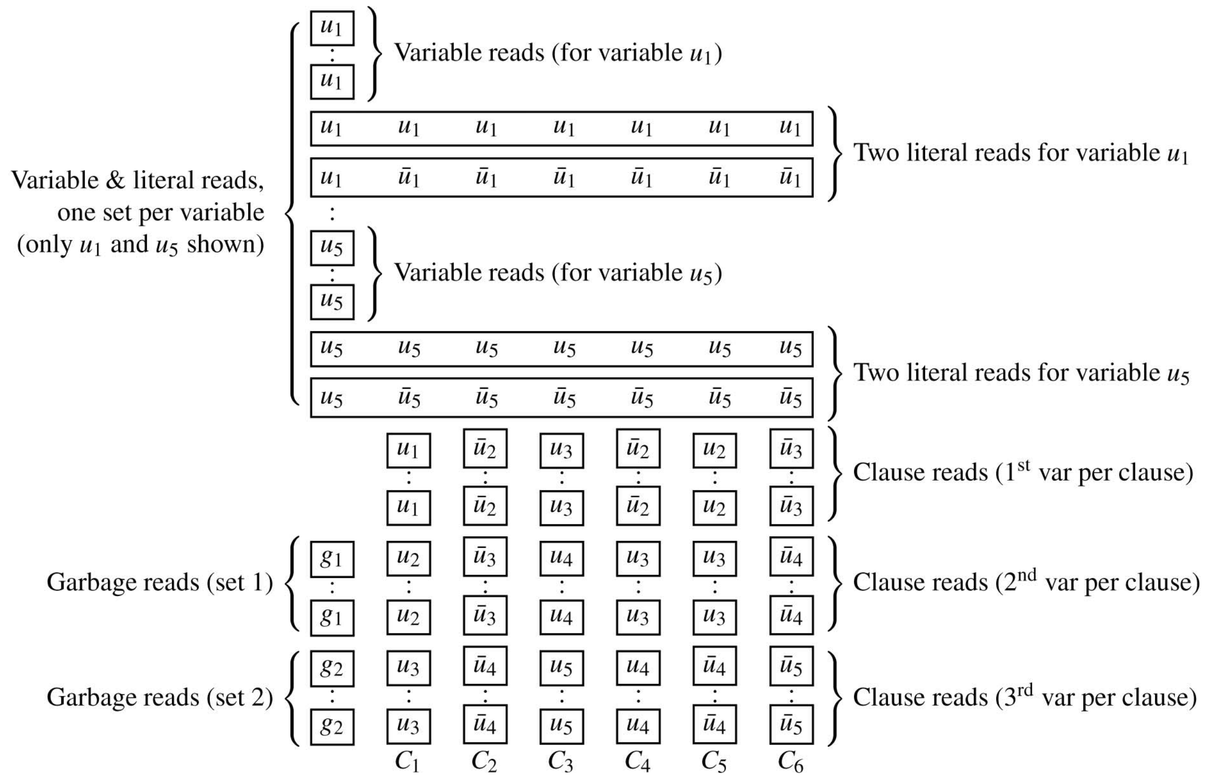
Given a set C of clauses over the variables of U, we construct in polynomial time a set R of reads over alphabet Σ, a position function p and a positive integer k such that C is satisfiable if and only if some k-set of superstrings is consistent with all but $|U|$ of the reads.

Letting $t = |U|$, we show how to construct a target problem instance with $k = t + 2$, with the intuition that a distinct superstring will correspond to each of the t variables, with two additional superstrings. This set of superstrings will be consistent with all but exactly t of the reads if, and only if, there is a satisfying truth assignment for the source instance of 3SAT. In case, the source problem has a satisfying truth assignment, the t reads that will not be consistent with any superstring will correspond to the negation of the literals of a satisfying truth assignment. The additional two superstrings can be thought of as accounting for the remaining two literals in each clause possibly not included in the satisfying truth assignment.

We first define the sequence alphabet Σ consisting of three subsets:

$$\Sigma = \Sigma_U \cup \Sigma_L \cup \Sigma_G.$$

We call Σ_U the *variable letters*, Σ_L the *literal letters* and Σ_G the *garbage-collecting letters*. The garbage-collecting letters are



Variable & literal reads, one set per variable (only u_1 and u_5 shown)
Variable reads (for variable u_1)
Two literal reads for variable u_1
Variable reads (for variable u_5)
Two literal reads for variable u_5
Clause reads (1st var per clause)
Garbage reads (set 1)
Clause reads (2nd var per clause)
Garbage reads (set 2)
Clause reads (3rd var per clause)
C_1 C_2 C_3 C_4 C_5 C_6

Fig. 5. Example transformation from 3SAT to MFR-RMA. The source problem instance is $U = \{u_1, u_2, u_3, u_4, u_5\}$ and $C = \{u_1u_2u_3, \bar{u}_2\bar{u}_3\bar{u}_4, u_3u_4u_5, \bar{u}_2u_3u_4, u_2u_3\bar{u}_4, \bar{u}_3\bar{u}_4\bar{u}_5\}$, six clauses on five variables. Boxed letters represent individual reads. Vertical dots separating identical reads indicate that those reads are repeated $|U| + 1$ times (in this case six times) at the same position.

simply $\Sigma_G = \{g_1, g_2\}$. The variable letters map bijectively to the variables of U:

$$\Sigma_U = \{u_1, \ldots, u_t\}.$$

The literal letters (a superset of the variable letters) consist of one letter for each literal:

$$\Sigma_L = \{u_1, \bar{u}_1, \ldots, u_t, \bar{u}_t\}.$$

The reads consist of the following four sets:

$$R = R_U \cup R_L \cup R_G \cup R_C.$$

We refer to R_U as the *variable reads*, R_L as the *literal reads*, R_C as the *clause reads*, and R_G are the *garbage-collecting reads*. All variable reads, garbage-collecting reads, and clause reads have length equal to 1 and all literal reads have length $|C| + 1$.

For each variable, we construct $t + 1$ reads in R_U, so $|R_U| = t(t + 1)$. For each variable read $r \in R_U, |r| = 1$ and consists of the letter from Σ_U corresponding to the same variable as r. For each variable read $r \in R_U$, we set $p(r) = (1, 1)$. As with the variable reads, the garbage-collecting reads, which number $2(t + 1)$ are each of length 1 and $t + 1$ of them consist of the letter g_1, while the remaining $t + 1$ consist of the letter g_2. For every $r \in R_G, p(r) = (1, 1)$.

The clause reads R_C also each consist of a single letter. For each clause c_i, and each literal in c_i, we create $(t + 1)$ clause reads consisting of the literal letter corresponding to that literal. For each read $r \in R_C$, if r corresponds to clause c_i then $p(r) = (i + 1, i + 1)$.

The $2t$ literal reads in R_L each have length $|C| + 1$, and begin with a variable letter from Σ_U, followed by $|C|$ literal letters from Σ_L. For all $r \in R_L, p(r) = (1, |C| + 1)$. Corresponding to each variable in $u_i \in U$, the set R_L contains a pair of reads, one corresponding to the positive form of the u_i and the other to the negative form of u_i:

$$u_iu_iu_iu_i \cdots u_i \quad \text{and} \quad u_i\bar{u}_i\bar{u}_i\bar{u}_i \cdots \bar{u}_i,$$

respectively. Note that for every variable read $r_i \in R_U$, there are exactly two literal reads $r_j, r_k \in R_L$ such that r_i is consistent with each of r_j and r_k, overlapping each at a single position, but r_j and r_k are not consistent with each other. A diagram to illustrate this transformation can be seen in Fig. 5.

The following proposition (with proof omitted) follows directly from the fact that each distinct variable, clause, and garbage-collecting read appears $t + 1$ times.

Proposition 3. *Suppose, S is a set of length $|C| + 1$ strings and $|S| = t + 2$. If all but t reads from $R_U \cup R_L \cup R_C \cup R_G$ are consistent with S, then all reads from R_U, R_C and R_G are consistent with S.*

Proposition 4. *Suppose, a set S of $t + 2$ superstrings is consistent with all but t reads from $R_U \cup R_L \cup R_C \cup R_G$. Then exactly t reads from R_L can be consistent with S, and all must correspond to distinct variables.*

Proof. Clearly, at least t reads from R_L must be consistent with S. Suppose, more than t reads from R_L are consistent with S. Then at most one superstring from S can be consistent with a garbage-collecting read, violating Proposition 3. Suppose, exactly t reads from R_L are consistent with S, but two of them correspond to the same variable. Note that the two literal reads are not consistent with each other. Then the two superstrings consistent with those two reads can be consistent with at most $t+1$ reads, which correspond to the same variable, from R_U. This requires the remaining reads from R_U and R_G, totaling $(t+1)(t+1)$ to be consistent with the remaining t superstrings. But this is a contradiction, since at most $t+1$ reads from $R_G \cup R_U$ can be consistent with the same superstring. □

Theorem 6. *Minimum fragment removal RMA is NP-hard when nesting of reads is allowed.*

Proof. Suppose, there is a truth assignment to variables of U that satisfies C. Then construct t superstrings identical through their entire sequence to the literal reads of R_L corresponding to the literals appearing in the truth assignment. These t superstrings will be consistent with every read from R_U and at least $t+1$ clause reads for every clause in C. Form two additional superstrings, each consistent with one of the garbage-collecting reads, and any of the clause reads corresponding to literals in the clauses that are not part of the truth assignment. This may leave gaps in these final two superstrings, which can be filled in arbitrarily. The only reads not consistent with at least one of these superstrings will be the t literal reads, corresponding to literals not included in the truth assignment.

Suppose, there is a set S of length $|C|+1$ superstrings consistent with all but t of the reads. Then, by Proposition 4, the t reads inconsistent with S must be literal reads, and those consistent with S correspond to a valid truth assignment. Let $s_{g_1}, s_{g_2} \in S$ be the superstrings consistent with reads from R_G. Since s_{g_1} and s_{g_2} can be consistent with at most $2(t+1)$ reads from R_C corresponding to each clause, the remaining reads from R_C for each clause must be consistent with one of the superstrings from $S \setminus \{s_{g_1}, s_{g_2}\}$, which are identical in sequence to the t literal reads consistent with S. The truth assignment for U corresponding to these literal reads will then satisfy each clause. □

We note also that the alphabet can be replaced with a binary alphabet simply by assigning a number to each letter of the alphabet used in our reduction, and replacing each letter in the reads with the binary representation of the associated number. Just as no two distinct letters from our alphabet matched each other, no two binary representations of numbers will match each other in every digit. We can also further restrict the problem by requiring that each of the $t+1$ copies of the variable reads, garbage collection reads, and clause reads be replaced by $t+1$ nonoverlapping, but consecutive copies. Correspondingly, the length of the literal reads would increase by a factor of $t+1$. Hence the hardness result holds for a binary alphabet and without duplicate reads at any position.

5 Discussion

We have described a set of computational problems in resolving distinct sequences from a heterogenous sample. We refer to these problems collectively as referenced multiple assembly problems. In these problems the distinct sequences are assumed sufficiently similar that all reads sequenced from the sample can map to a common reference genome. The reference genome provides knowledge of positions for reads. These problems emerge in a wide variety of contexts, including, but not limited to, haplotyping inference for polyploid species and metagenomic data analysis.

We described an algorithm for the RMA problem that finds a minimum set of sequences that is consistent with all reads in $O(N)$ time, which is linear in the size of the input reads and a significant improvement over a previously used algorithm for the same problem [10]. Although our algorithm does not consider sequencing errors, we remark that error rates in sequencing are falling rapidly. For current applications, the algorithm can be applied following an error-correction stage as is common in sequence assembly.

We also examined the MFR-RMA problem, which aims to assemble a set of k sequences that is consistent with the largest subset of reads. We showed that MFR-RMA can be solved efficiently by employing an elegant method of Andras Frank originally designed to find optimal sets of chains and antichains of posets [17]. The MFR-RMA variant is particularly suitable when the set of reads may have contamination, or when the number of sequences to assemble is known a priori (e.g., haplotype inference for polyploids of known ploidy). It is likely that improvements to the time complexity of the MFR-RMA variant can be obtained by exploiting the special structure of the problem rather than applying the general reduction to min-cost flow.

We also described two problem variants, MD-RMA and BD-RMA, that explicitly account for sequencing errors. While these two problems address the practical issue of errors in reads, they are easily seen to generalize other problems already shown to be NP-hard. The intractability seems to stem generally from allowing mismatches between the reads and their associated superstrings. We have not explored approximation algorithms for the problem variants found to be NP-hard. One interesting avenue for future algorithmic work includes analysis of the parameterized complexity of these problems. These problems may be tractable if some of their many natural parameters are fixed (e.g., length of reads, numbers of superstrings, alphabet size, etc.). For example, in second-generation sequencing applications the initial mapping stage often restricts the number of possible errors in the reads, and allow only a fixed and relatively small number of mismatches between reads and the reference genome. Another direction for future research with practical implications will be to investigate the complexity of these problems for paired-end reads.

Acknowlegments

The authors would like to thank Michael Waterman and Vineet Bafna for helpful discussions.

References

[1] V. Bafna, D. Gusfield, G. Lancia, and S. Yooseph, "Haplotyping as Perfect Phylogeny: A Direct Approach," *J. Computational Biology*, vol. 10, nos. 3/4, pp. 323-340, 2003.

[2] V. Bafna, S. Istrail, G. Lancia, and R. Rizzi, "Polynomial and APX-Hard Cases of the Individual Haplotyping Problem," *Theoretical Computer Science*, vol. 335, no. 1, pp. 109-125, 2005.

[3] V. Bansal, A.L. Halpern, N. Axelrod, and V. Bafna, "An MCMC Algorithm for Haplotype Assembly from Whole-Genome Sequence Data," *Genome Research,* vol. 18, no. 8, pp. 1336-1346, 2008.

[4] L.G. Biesecker et al., "The Clinseq Project: Piloting Large-Scale Genome Sequencing for Research in Genomic Medicine," *Genome Research,* vol. 19, no. 9, pp. 1665-1674, 2009.

[5] R. Cilibrasi, L. van Iersel, S. Kelk, and J. Tromp, "The Complexity of the Single Individual SNP Haplotyping Problem," *Algorithmica,* vol. 49, no. 1, pp. 13-36, Sept. 2007.

[6] S.A. Cook and R.A. Reckhow, "Time Bounded Random Access Machines," *J. Computer and System Sciences,* vol. 7, no. 4, pp. 354-375, 1973.

[7] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms.* MIT Press, 2001.

[8] R.P. Dilworth, "A Decomposition Theorem for Partially Ordered Sets," *The Annals of Math.,* vol. 51, no. 1, pp. 161-166, 1950.

[9] J. Edmonds and R.M. Karp, "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems," *J. ACM,* vol. 19, no. 2, pp. 248-264, 1972.

[10] N. Eriksson, L. Pachter, Y. Mitsuya, S.-Y. Rhee, C. Wang, B. Gharizadeh, M. Ronaghi, R.W. Shafer, and N. Beerenwinkel, "Viral Population Estimation Using Pyrosequencing," *PLoS Computational Biology,* vol. 4, no. 5, p. e1000074, May 2008.

[11] E. Eskin, E. Halperin, and R. Karp, "Efficient Reconstruction of Haplotype Structure via Perfect Phylogeny," *J. Bioinformatics Computational Biology,* vol. 1, no. 1, pp. 1-20, 2003.

[12] S. Felsner, V. Raghavan, and J. Spinrad, "Recognition Algorithms for Orders of Small Width and Graphs of Small Dilworth Number," *Order,* vol. 20, no. 4, pp. 351-364, Nov. 2003.

[13] L.R. Ford and D.R. Fulkerson, "Maximal Flow through a Network," *Canadian J. Math.,* vol. 8, no. 3, pp. 399-404, 1956.

[14] L.R. Ford and D.R. Fulkerson, "Constructing Maximal Dynamic Flows from Static Flows," *Operations Research,* vol. 6, no. 3, pp. 419-433, 1958.

[15] L.R. Ford and D.R. Fulkerson, *Flows in Networks.* Princeton Univ. Press, 1962.

[16] M. Frances and A. Litman, "On Covering Problems of Codes," *Theory of Computing Systems,* vol. 30, no. 2, pp. 113-119, Mar. 1997.

[17] A. Frank, "On Chain and Antichain Families of a Partially Ordered Set," *J. Combinatorial Theory, Series B,* vol. 29, no. 2, pp. 176-184, Oct. 1980.

[18] K.A. Frazer et al., "A Second Generation Human Haplotype Map of over 3.1 Million SNPs," *Nature,* vol. 449, no. 7164, pp. 851-861, 2007.

[19] E. Fredkin, "Trie Memory," *Comm. ACM,* vol. 3, no. 9, pp. 490-499, 1960.

[20] D.R. Fulkerson, "Note on Dilworth's Decomposition Theorem for Partially Ordered Sets," *Proc. Am. Math. Soc.,* vol. 7, no. 4, pp. 701-702, Aug. 1956.

[21] C.W. Fuller et al., "The Challenges of Sequencing by Synthesis," *Nature Biotechnology,* vol. 27, no. 11, pp. 1013-1023, 2009.

[22] H.N. Gabow and R.E. Tarjan, "Faster Scaling Algorithms for Network Problems," *SIAM J. Computing,* vol. 18, pp. 1013-1036, 1989.

[23] S.R. Gill, M. Pop, R.T. DeBoy, P.B. Eckburg, P.J. Turnbaugh, B.S. Samuel, J.I. Gordon, D.A. Relman, C.M. Fraser-Liggett, and K.E. Nelson, "Metagenomic Analysis of the Human Distal Gut Microbiome," *Science,* vol. 312, no. 5778, pp. 1355-1359, 2006.

[24] C. Green and D. Kleitman, "The Structure of Sperner K-Family," *J. Combinatorial Theory (A),* vol. 20, pp. 80-88, 1976.

[25] C. Greene, "Some Partitions Associated with a Partially Ordered Set," *J. Combinatorial Theory (A),* vol. 20, no. 1, pp. 69-79, 1976.

[26] D. Gusfield, "Haplotyping as Perfect Phylogeny: Conceptual Framework and Efficient Solutions," *Proc. Sixth Ann. Int'l Conf. Computational Biology (RECOMB '02),* pp. 166-175, 2002.

[27] M. Jakobsson, S.W. Scholz, P. Scheet, J.R. Gibbs, J.M. VanLiere, H.-C. Fung, Z.A. Szpiech, J.H. Degnan, K. Wang, R. Guerreiro, J.M. Bras, J.C. Schymick, D.G. Hernandez, B.J. Traynor, J. Simon-Sanchez, M. Matarin, A. Britton, J. van de Leemput, I. Rafferty, M. Bucan, H.M. Cann, J.A. Hardy, N.A. Rosenberg, and A.B. Singleton, "Genotype, Haplotype and Copy-Number Variation in Worldwide Human Populations," *Nature,* vol. 451, no. 7181, pp. 998-1003, 2008.

[28] J.Y. Kim, S. Tavaré, and D. Shibata, "Counting Human Somatic Cell Replications: Methylation Mirrors Endometrial Stem Cell Divisions," *Proc. Nat'l Academy of Sciences USA,* vol. 102, no. 49, pp. 17739-17744, 2005.

[29] H.R. Kobel and L. Du Pasquier, "Genetics of Polyploid Xenopus," *Trends in Genetics,* vol. 2, pp. 310-315, 1986.

[30] P.W. Laird, "The Power and the Promise of DNA Methylation Markers," *Nature Rev. Cancer,* vol. 3, no. 4, pp. 253-266, 2003.

[31] G. Lancia, V. Bafna, S. Istrail, R. Lippert, and R. Schwartz, "SNPs Problems, Complexity and Algorithms," *Proc. Ann. European Symp. Algorithms (ESA),* F.M. auf der Heide, ed., pp. 182-193, 2001.

[32] S. Levy, G. Sutton, P.C. Ng, L. Feuk, A.L. Halpern, B.P. Walenz, N. Axelrod, J. Huang, E.F. Kirkness, G. Denisov, Y. Lin, J.R. MacDonald, A.W.C. Pang, M. Shago, T.B. Stockwell, A. Tsiamouri, V. Bafna, V. Bansal, S.A. Kravitz, D.A. Busam, K.Y. Beeson, T.C. McIntosh, K.A. Remington, J.F. Abril, J. Gill, J. Borman, Y.-H. Rogers, M.E. Frazier, S.W. Scherer, R.L. Strausberg, and J.C. Venter, "The Diploid Genome Sequence of an Individual Human," *PLoS Biology,* vol. 5, no. 10, p. e254, Sept. 2007.

[33] L.M. Li, J.H. Kim, and M.S. Waterman, "Haplotype Reconstruction from SNP Alignment," *J. Computational Biology,* vol. 11, nos. 2/3, pp. 505-516, 2004.

[34] A. Ludwig, N.M. Belfiore, C. Pitra, V. Svirsky, and I. Jenneckens, "Genome Duplication Events and Functional Reduction of Ploidy Levels in Sturgeon (Acipenser, Huso and Scaphirhynchus)," *Genetics,* vol. 158, no. 3, pp. 1203-1215, 2001.

[35] L.A. Meyers and D.A. Levin, "On the Abundance of Polyploids in Flowering Plants," *Evolution,* vol. 60, no. 6, pp. 1198-1206, 2006.

[36] A. Mortazavi, B.A. Williams, K. McCue, L. Schaeffer, and B. Wold, "Mapping and Quantifying Mammalian Transcriptomes by RNA-Seq," *Nature Methods,* vol. 5, no. 7, pp. 621-628, 2008.

[37] S. Ohno, *Evolution by Gene Duplication.* Springer-Verlag, 1970.

[38] P.A. Pevzner, H. Tang, and M.S. Waterman, "An Eulerian Path Approach to DNA Fragment Assembly," *Proc. Nat'l Academy of Sciences USA,* vol. 98, no. 17, pp. 9748-9753, 2001.

[39] D.E. Schones and K. Zhao, "Genome-Wide Approaches to Studying Chromatin Modifications," *Nature Rev. Genetics,* vol. 9, no. 3, pp. 179-191, 2008.

[40] D.C. Schwartz and M.S. Waterman, "New Generations: Sequencing Machines and Their Computational Challenges," *J. Computer Science and Technology,* vol. 25, no. 1, pp. 3-9, 2010.

[41] J. Shendure and H. Ji, "Next-Generation DNA Sequencing," *Nature Biotechnology,* vol. 26, no. 10, pp. 1135-1145, Oct. 2008.

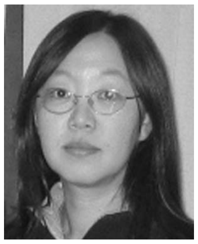
[42] D. Shibata and S. Tavaré, "Counting Divisions in a Human Somatic Cell Tree: How, What and Why," *Cell Cycle,* vol. 5, no. 6, pp. 610-614, 2006.

[43] S.G. Tringe and E.M. Rubin, "Metagenomics: DNA Sequencing of Environmental Samples," *Nature Rev. Genetics,* vol. 6, no. 11, pp. 805-814, 2005.

[44] S.G. Tringe et al., "Comparative Metagenomics of Microbial Communities," *Science,* vol. 308, no. 5721, pp. 554-557, 2005.

[45] J.A. Udall and J.F. Wendel, "Polyploidy and Crop Improvement," *Crop Science,* vol. 46, no. 1, pp. S3-S14, 2006.

[46] Y. Yatabe, S. Tavaré, and D. Shibata, "Investigating Stem Cells in Human Colon by Using Methylation Patterns," *Proc. Nat'l Academy of Sciences USA,* vol. 98, no. 19, pp. 10839-10844, 2001.



Qian Peng received the BS degree in computer science from the Beijing University and the MS in biomedical engineering from the University of Tennessee, Memphis, and currently she is working toward the PhD degree in the Department of Computer Science and Engineering, University of California, San Diego. Her research interests are computational biology and bioinformatics.

Andrew D. Smith received the BA degree in psychology, the bachelor of computer science (BCS) degree in 2000, and the PhD degree in computer science in 2004 from the University of New Brunswick. He studied computational biology and genomics in Cold Spring Harbor Laboratory until 2008 at which time he moved to University of Southern California where he is currently an assistant professor of biological sciences.