Decoding Genetic Variations: Communications-Inspired Haplotype Assembly

Zrinka Puljiz and Haris Vikalo

Abstract—High-throughput DNA sequencing technologies allow fast and affordable sequencing of individual genomes and thus enable unprecedented studies of genetic variations. Information about variations in the genome of an individual is provided by haplotypes, ordered collections of single nucleotide polymorphisms. Knowledge of haplotypes is instrumental in finding genes associated with diseases, drug development, and evolutionary studies. Haplotype assembly from high-throughput sequencing data is challenging due to errors and limited lengths of sequencing reads. The key observation made in this paper is that the minimum error-correction formulation of the haplotype assembly problem is identical to the task of deciphering a coded message received over a noisy channel—a classical problem in the mature field of communication theory. Exploiting this connection, we develop novel haplotype assembly schemes that rely on the bit-flipping and belief propagation algorithms often used in communication systems. The latter algorithm is then adapted to the haplotype assembly of polyploids. We demonstrate on both simulated and experimental data that the proposed algorithms compare favorably with state-of-the-art haplotype assembly methods in terms of accuracy, while being scalable and computationally efficient.

Index Terms—Haplotype assembly, belief propagation, bit flipping

1 Introduction

Recent advancements in high-throughput DNA sequencing [1], [2], [3], [4] have enabled fast and affordable re-sequencing of individual genomes and hence opened up the possibility of conducting routine tests of genetic variations. Identification and study of such variations helps reveal susceptibility to genetic and complex diseases, and may lead to the development of personalized treatment plans adjusted to individual genetic codes [5], [6], [7], [8]. Majority of chromosome pairs in diploid organisms, including humans, are homologous— they carry fundamentally the same type of information and are structurally similar but not identical. The most common type of variation between chromosomes in a homologous pair are single nucleotide polymorphisms (SNPs), i.e., occurrence of different nucleotides in the corresponding locations on the chromosomes. Variations between chromosomes are fully specified by haplotypes, ordered sequences of SNPs associated with each of the chromosomes.

To assemble haplotypes of an individual organism, we may rely on high-throughput DNA sequencing platforms which oversample the DNA sequence to create a library of overlapping reads. The reads are relatively short – typically, on the order of hundreds of nucleotides. Paired-end reads link genome fragments that are large distances apart (hundreds to thousands of bases) by having inserts of approximately known lengths that separate two reads. To provide information that can be used to assemble haplotypes, a pair of linked reads must cover more than one SNP location. If sequencing were not affected by errors and the subsequent SNP and genotype calling steps were free of any uncertainties, haplotype assembly for diploid organisms would be straightforward and could be reduced to separating reads into two subgroups—one for each haplotype in a pair. In the realistic case of having erroneous data, such classification necessarily results in subgroups that contain reads with conflicting information. In literature, haplotype assembly has led to several optimization problems, most of them attempting to minimize the number of transformations of the data set needed to make the reads consistent with having originated from one of the chromosomes [9]. In particular, it motivated the use of the minimum fragment removal, minimum edge removal, minimum SNP removal, and minimum error correction (MEC) optimization criteria. The MEC criterion, which attracted the most attention in recent years and is known to be NP-hard [10], is considered in this paper. Various methods for doing haplotype assembly by optimizing the MEC criterion have been developed. The optimal yet computationally intensive branch-and-bound scheme was proposed in [11]. As an alternative, several heuristic algorithms that trade off accuracy for speed have been proposed [11], [12], [13], [14], [15], [16], [17]. More recent methods include HapCompass [18] and the widely used HapCut [19] algorithm.

Our key observation presented in this paper has been that the MEC formulation of the haplotype assembly problem is identical to the task of deciphering a coded message transmitted over a noisy communication channel [20], [21]. Decoding of noisy messages has been extensively studied in communication theory over the last several decades [22], [23], [24], [25], [26]. Exploiting the aforementioned connection, we first propose a haplotype assembly method that relies on the bit-flipping (BF) algorithm originally developed in the context of decoding low-density parity check (LDPC) codes [22]. We then design a belief propagation (BP) algorithm that provides higher accuracy than the

• *The authors are with the Department of Electrical and Computer Engineering, University of Texas in Austin, Austin, TX, 78712.*
E-mail: zrinka@utexas.edu, hvikalo@ece.utexas.edu.

Manuscript received 18 Sept. 2014; revised 14 Mar. 2015; accepted 20 July 2015. Date of publication 29 July 2015; date of current version 1 June 2016.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TCBB.2015.2462367

bit-flipping scheme at the cost of a slight increase in computational complexity. When tested on the *1,000 Genomes Project* and Fosmid [27] data sets, the proposed methods compare favorably with HapCut and HapCompass while being significantly faster.

Beside diploids, high-throughput sequencing has enabled studies of genetic variations in polyploid organisms which have $k > 2$ chromosomes. Haplotype assembly for polyploids is considerably more challenging and requires larger coverage to enable separation of the reads. Existing prior work on haplotype assembly for polyploids includes HapCompass [18] and HapTree [28]. We extend our belief propagation algorithm to the assembly of polyploid haplotypes and demonstrate in simulation studies that it significantly outperform HapCompass.

The remainder of the paper is organized as follows. We start by considering haplotype assembly for diploids and introduce the system model and problem formulation in Section 2. In Sections 3 and 4, we present the main contributions of the paper: a representation of haplotype assembly as a decoding problem and two algorithms that rely on that representation, respectively. An analytical bound on the performance is given in Section 5. Section 6 extends the belief propagation algorithm to the polyploid case. Results and discussion are presented in Section 7 while the conclusion and future work are in Section 8.

2 Notation and Problem Statement

Following sequencing, aligning to a reference, and SNP and genotype calling, to facilitate haplotype assembly the data is typically organized in a SNP fragment matrix. Segments of the reads that cover homozygous sites provide no information about haplotypes and are hence discarded. Since the SNP sites in diploids are typically bi-allelic, we may denote them using binary symbols $\{0, 1\}$. The (i, j) entry of the SNP fragment matrix $\mathbf{R}$ indicates the information about the jth SNP site provided by the ith read; if the ith read does not cover the jth SNP site, the (i, j) entry of $\mathbf{R}$ is denoted by the symbol $\times$. As an illustration, the resulting SNP fragment matrix may have the following form,

$$\mathbf{R} = \begin{bmatrix} \times & \times & 0 & \times & \times & 1 \\ \times & 1 & \times & \times & 0 & \times \\ \times & \times & 0 & \times & 0 & \times \\ 0 & \times & \times & 1 & \times & \times \\ 1 & \times & 1 & \times & \times & \times \\ \times & \times & 1 & \times & 0 & \times \\ \times & 0 & \times & 0 & \times & \times \\ \times & \times & \times & 0 & \times & 0 \end{bmatrix}.$$

2.1 Notation

Throughout the paper, we use the following notation:

$(\mathbf{h}, \bar{\mathbf{h}})$: an unordered pair of a haplotype and its complement, with support in $\{0, 1\}^n$.

$(\mathbf{h}_0, \mathbf{h}_1, \ldots, \mathbf{h}_{k-1})$: an unordered k-touple of all the haplotypes on all the chromosomes with support in $\{0, 1\}^n$.

$\mathbf{r}_i$: read i, $1 \le i \le m$, with support in $\{0, 1, \times\}^n$, where $\times$ denotes unobserved alleles.

$\mathbf{R}$: an $n \times m$ matrix whose ith row corresponds to $\mathbf{r}_i$ and jth column corresponds to the jth SNP site.

$\mathbf{s}$: a vector indicating whether a read is associated with $\mathbf{h}_0$, $\mathbf{h}_1$, … or $\mathbf{h}_{k-1}$, with support in $\{0, 1, \ldots, k-1\}^m$.

$\mathbf{c}$: a vector collecting numeric entries of the matrix $\mathbf{R}$.

$\mathbf{G}$: generator matrix of a linear block code.

$\mathbf{H}$: parity-check matrix associated with the generator matrix $\mathbf{G}$.

2.2 Problem Statement

Define a measure of distance d between two symbols in the ternary alphabet $\{0, 1, \times\}$ as

$$d(a, b) = \begin{cases} |a - b| & \text{if } a \neq \times \text{ and } b \neq \times, \\ 0, & \text{otherwise.} \end{cases}$$

With the adopted MEC criterion, the goal of haplotype assembly for diploid organisms is to minimize Z over a binary vector $\mathbf{h}$,

$$Z = \sum_{i=1}^{m} \min(\mathrm{hd}(\mathbf{r}_i, \mathbf{h}), \mathrm{hd}(\mathbf{r}_i, \bar{\mathbf{h}})), \tag{1}$$

where $\mathrm{hd}(\cdot, \cdot)$ denotes the generalized Hamming distance defined as

$$\mathrm{hd}(\mathbf{r}_i, \mathbf{h}) = \sum_{j=1}^{n} d(R(i, j), h_j),$$

where $R(i, j)$ denotes the (i, j) entry in $\mathbf{R}$. Intuitively, minimizing (1) leads to finding the smallest number of binary entries (alleles) in $\mathbf{R}$ that should be flipped so that the rows of $\mathbf{R}$ can be unambiguously associated with either $\mathbf{h}$ or $\bar{\mathbf{h}}$. For convenience, we will refer to $\mathbf{h}$ as the reference haplotype.

The MEC objective is often used as a proxy for the switch error rate (SWER). The switch between positions i and $i+1$ is defined as the event $\hat{\mathbf{h}}_i = \mathbf{h}_i$ and $\hat{\mathbf{h}}_{i+1} \neq \mathbf{h}_{i+1}$, or $\hat{\mathbf{h}}_i \neq \mathbf{h}_i$ and $\hat{\mathbf{h}}_{i+1} = \mathbf{h}_{i+1}$. Note that, unlike the MEC score that is a function of the fragment matrix and the recovered haplotype, SWER is calculated with respect to a (known) underlying haplotype. We will use SWER to test the performance of the proposed algorithms in Section 7.

3 Reformulating Haplotype Assembly as the Decoding Problem

It is beneficial to briefly consider the scenario where the SNP fragment matrix $\mathbf{R}$ is error-free. In particular, let us assume that the reference haplotype

$$\mathbf{h} = [0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1]$$

and its complement $\bar{\mathbf{h}}$ (each comprising $m = 6$ alleles) were sampled with $n = 8$ reads, and that the origin of the reads is indicated in the following read select vector,

$$\mathbf{s} = [0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 1]^T.$$

If the value of the ith component of $\mathbf{s}$ is 0, that indicates the ith row of $\mathbf{R}$ originated by sampling $\mathbf{h}$; otherwise, it originated by sampling $\bar{\mathbf{h}}$. Then the corresponding error-free fragment matrix is of the form

TABLE 1
Tabular Representation of the Known Entries in the Error-Free Fragment SNP Matrix as a Function of the Reference Haplotype and Read Select Variables

s_i	h_j	$R_{i,j}$
0	0	0
0	1	1
1	0	1
1	1	0

$$\mathbf{R} = \begin{bmatrix} \times & \times & 0 & \times & \times & 1 \\ \times & 1 & \times & \times & 0 & \times \\ \times & \times & 0 & \times & 0 & \times \\ 0 & \times & \times & 1 & \times & \times \\ 1 & \times & 1 & \times & \times & \times \\ \times & \times & 1 & \times & 1 & \times \\ \times & 0 & \times & 0 & \times & \times \\ \times & \times & \times & 0 & \times & 0 \end{bmatrix}.$$

Clearly, all of the variables in the previous representation of the data (**h**, **s**, and **R**) have binary numerical entries. A closer inspection reveals that if the (i, j) entry in **R**, $R_{i,j}$, is numerical (i.e., 0 or 1), it is obtained as the result of an exclusive-OR (XOR) operation between the ith and jth entries of **s** and **h**, respectively, as indicated in Table 1.

Interestingly, XOR functions are building blocks of error-correcting codes in communication systems, which we briefly summarize next.

3.1 Communication Systems

In data communication systems, the goal of point-to-point communication is to reliably transmit and receive (decode) messages that are adversely affected by the transmission medium. The most simple communication system consisting of a source, encoder, channel, decoder and destination is illustrated in Fig. 1. The coder introduces redundancy into the message in order to combat unknown effects of the noisy channel by adding redundancy to the binary messages generated by the source. Output of the coder—a codeword that belongs to a predefined codebook—is transmitted across the channel that modifies it in a nondeterministic way. The task of the decoder is to reverse the effects of the channel and map the received signal back to the "closest" valid codeword, which is then converted back into the message and forwarded to the destination.

A special class of codes are those where each bit in the codeword is a linear function of the message bits. Such linear codes are fully described by a code generator matrix **G**. Each column of **G** specifies the linear function used to obtain the corresponding codeword bit, and the entire codeword is formed by simply multiplying over $GF(2)$ the message with the code generator matrix; multiplications over $GF(2)$ are identical to the exclusive-OR operations illustrated in Table 1. We remark that a subclass of communication channels, so-called binary symmetric channels (BSCs), invert each transmitted bit independently with the same probability.

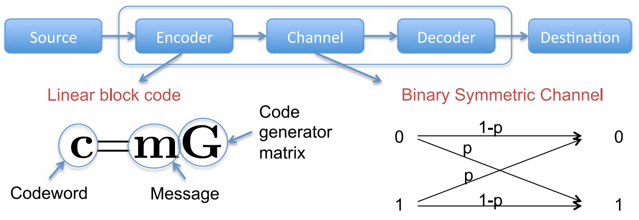

Fig. 1. Components of a simple communication system operate as follows: 1) a message is sent by a source, 2) a coder maps messages to a codeword using a set of linear functions, 3) the codeword is corrupted by the binary symmetric channel, 4) a decoder maps back the corrupted codeword into a valid message, and 5) the recovered message reaches the destination.

3.2 Decoding Haplotypes

Let us define a "message" as the vector formed by concatenating the haplotype vector **h** with the read select vector **s**, $\mathbf{m} = [\mathbf{h} \quad \mathbf{s}]$. Let $\{f_k\}$ denote the collection of indices $\{(i_k, j_k)\}$ identifying positions where the matrix **R** has numeric entries, i.e., $R(i_k, j_k) \neq \times$, $1 \leq k \leq M$, where M denotes the total number of informative (binary) entries in **R**. Define the "code generating" matrix **G** with the entries

$$G(l, k) = \begin{cases} 1 \text{ if } l = j_k \text{ or } l = i_k + n, \ 1 \leq k \leq M, \\ 0, \text{ otherwise}, \end{cases}$$

where n denotes the length of the haplotype. Therefore, each column of **G** by construction has only two non-zero elements, one at the location (j_k, k) and the other at the location $(n + i_k, k)$. To clarify the construction of **G**, we give an example next.

Example 1. Consider the following SNP fragment matrix,

$$\mathbf{R} = \begin{bmatrix} 0 & 1 & \times \\ \times & 1 & 1 \\ 1 & \times & 0 \end{bmatrix}.$$

The set $\{f_k\}$ collects indices of the binary elements in **R** in a row-wise order, $\{f_k\} = \{(1, 1), (1, 2), (2, 2), (2, 3), (3, 1), (3, 3)\}$. The corresponding code generating matrix **G** has the following form,

$$\mathbf{G} = \left[\begin{array}{cccccc} 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ \hline 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{array}\right].$$

Each column of **G** is associated with one binary entry in **R**, where the two non-zero entries in the kth column of **G** correspond to indices (i_k, j_k) of the kth (ordered row-wise) informative entry in **R**, $R(i_k, j_k) \neq \times$. The horizontal line in **G** separates the rows of **G** associated with the columns i_k of **R** (i.e., with the SNP position) from the rows of **G** associated with the rows j_k of **R** (i.e., with the reads).

In the absence of SNP-calling errors, it holds that

$$\mathbf{c} = [\mathbf{h} \quad \mathbf{s}]\mathbf{G}, \tag{2}$$

where the kth entry of **c** is equal to the kth (ordered row-wise) numeric entry of **R**, i.e., $c_k = R(i_k, j_k)$.

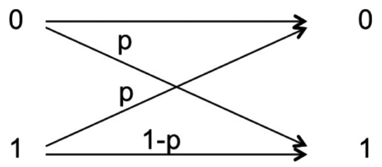

Fig. 2. An illustration of a binary symmetric channel with crossover probability p.

Remark 1. Note that any permutation of the rows or columns of $\mathbf{G}$ corresponds to the permutation of the components in the vectors $\mathbf{h}$ and $\mathbf{s}$, or the order in which numeric entries of matrix $\mathbf{R}$ occur in the codeword $\mathbf{c}$. All such generated code generator matrices are valid (and equivalent) representations of $\mathbf{R}$.

Going back to the setting of Example 1 where $\mathbf{h} = [0\ 1\ 1]$, $\mathbf{s} = [0\ 0\ 1]$ and $\mathbf{c} = [0\ 1\ 1\ 1\ 1\ 0]$, it is easy to verify (2). However, sequencing errors adversely affect SNP and genotype calling and hence the SNP fragment matrix $\mathbf{R}$ typically has a fraction of entries that are incorrect (flipped). This can be modeled by thinking of the "codeword" $\mathbf{c}$ in (2) as being transmitted across the binary symmetric channel illustrated in Fig. 2, where p denotes the probability of inverting a bit (i.e., p represents the error rate in $\mathbf{R}$). Therefore, the possibly erroneous entries in $\mathbf{R}$ can be represented as

$$\mathbf{y} = [\mathbf{h} \quad \mathbf{s}]\mathbf{G} + \mathbf{e}, \tag{3}$$

where the kth entry of $\mathbf{y}$ is $y_k = R(i_k, j_k)$, $\mathbf{e}$ denotes the error vector, and all the operations are in $GF(2)$. The goal of haplotype assembly can be restated as follows: *Given the vector* $\mathbf{y}$ *and the matrix* $\mathbf{G}$, *both derived from the SNP-fragment matrix* $\mathbf{R}$, *find the most likely vector* $[\mathbf{h} \quad \mathbf{s}]$.

To facilitate the decoding of $[\mathbf{h} \quad \mathbf{s}]$ and hence perform haplotype assembly, we rely on the parity check matrix $\mathbf{H}$. For the linear codes defined by the encoding operation (2), $\mathbf{H}$ is orthogonal to $\mathbf{G}^T$, i.e., the range of $\mathbf{G}^T$ is the null space of $\mathbf{H}$. Given $\mathbf{G}$, we find $\mathbf{H}$ by means of the simple Gaussian elimination. Note that

$$\mathbf{Hy} = \mathbf{H}\Big(\mathbf{G}^T[\mathbf{h} \quad \mathbf{r}]^T + \mathbf{e}\Big) = \mathbf{He},$$

and that the vector $\mathbf{e}$ can be viewed as the distance between the observations $\mathbf{y}$ and the closest valid codeword. The minimum distance decoding is concerned with finding the codeword $\mathbf{c}$ (or, equivalently, $\mathbf{e}$) that solves the minimization problem

$$\min_{\mathbf{c}\in\mathcal{C}} d(\mathbf{c},\mathbf{y}) = \min_{\mathbf{c}\in\mathcal{C}} \|\mathbf{y}-\mathbf{c}\|_0 = \min_{\mathbf{c}\in\mathcal{C},\mathbf{y}=\mathbf{c}+\mathbf{e}} \|\mathbf{e}\|_0 = \min_{\mathbf{e}:\mathbf{H}(\mathbf{y}+\mathbf{e})=0} \|\mathbf{e}\|_0, \tag{4}$$

where $\|\cdot\|_0$ denotes the l_0-norm (the number of non-zero entries) of its argument. Recall that the goal of the minimum error correction formulation of the haplotype assembly problem is to select $\mathbf{h}$ minimizing the number of entries in $\mathbf{R}$ that need to be flipped so that there is no conflicting information in the SNP-fragment matrix. Therefore, its objective (1) can be restated as

$$Z = \min_{\mathbf{e}:\mathbf{H}(\mathbf{y}+\mathbf{e})=0} \|\mathbf{e}\|_0.$$

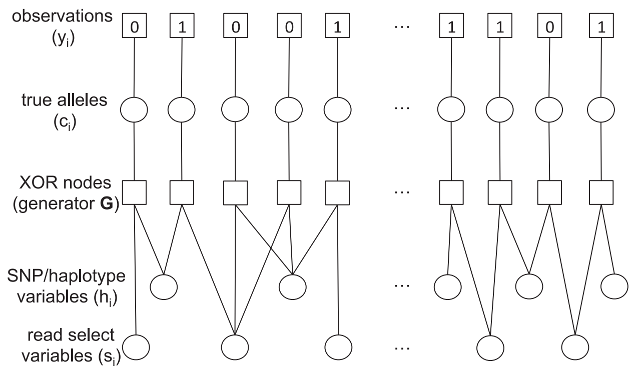

Fig. 3. A graphical model illustrating how the data used for haplotype assembly is generated. The numeric entries in the SNP fragment matrix $\mathbf{R}$ are collected into a vector $\mathbf{y}$; due to sequencing and data processing errors, these may differ from the true alleles. Read select variables (components of $\mathbf{s}$), SNP variables (components of the haplotype vector $\mathbf{h}$), and true alleles (components of $\mathbf{c}$) are connected through check nodes (i.e., XOR functions defined by the structure of $\mathbf{G}$).

Comparing the two optimizations above, we see that the minimum distance decoding also leads to minimization of the MEC score. Moreover, note that in the absence of any prior information the minimum distance decoding coincides with the maximum a posteriori decoding.

Based on the observed connection between communications and haplotype assembly, we design graphical models and utilize them for the design of algorithms solving the latter problem.

3.3 Graphical Models

Fig. 3 shows the graphical representation of the measurement model (3), illustrating the interactions between elements of the read select vector $\mathbf{s}$, reference haplotype $\mathbf{h}$ and the observations collected in $\mathbf{y}$. Both the reference haplotype and read select vector are unobserved variables in this model, and the interactions between them are driven by the structure of the code generating matrix $\mathbf{G}$. These interactions are depicted by the square nodes in the graph, each connected to exactly one SNP variable (a component of $\mathbf{h}$), one read select variable (a component of $\mathbf{s}$) and one observation (a component of $\mathbf{y}$, connected via the codeword $\mathbf{c}$). This graphical model will be the basis for the derivation and implementation of the belief propagation algorithm for haplotype assembly presented in the next section.

An alternative graphical model, based on the parity check matrix $\mathbf{H}$, is illustrated in Fig. 4. The variables in this model

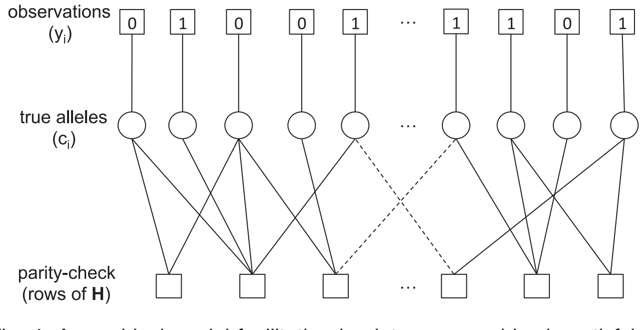

Fig. 4. A graphical model facilitating haplotype assembly via satisfying conditions imposed by the parity check matrix $\mathbf{H}$. The ith parity check node, defined by the ith row of $\mathbf{H}$, is connected to the variable c_k if $H(l,k) \neq 0$.

are associated with the entries in the vector **y**. The rows of the parity check matrix define parity check nodes illustrated in the figure. The edges in the graphical model emanating from a parity check node connect to the variables identified by the locations of the non-zero entries of the corresponding row of **H**. When the variables connected to the parity check nodes are such that their linear combinations (over GF(2)) at each node are zero, consistent haplotype can be recovered. This will be exploited to design and implement the bit-flipping haplotype assembly algorithm in the next section.

4 Haplotype Assembly via Decoding of Linear Block Codes

Motivated by the decoding algorithms that correct noise-induced errors in communication systems, in this section we present two haplotype assembly methods: the bit flipping algorithm and the belief propagation algorithm. These are computationally efficient yet highly accurate heuristics for solving the NP-hard assembly problem.

4.1 The Bit-Flipping Algorithm

The bit-flipping algorithm relies on the graphical model illustrated in Fig. 4. The basic idea of the algorithm is to examine each variable node of the graph and find the number of parity check equations $\mathbf{Hy} = 0$ it violates; there exists one such variable for each numerical entry in **R**. The bit in **y** with the largest margin of unsatisfied parity check equations versus satisfied parity check equations is flipped (i.e., the corresponding entry in **y** is changed from 0 to 1 or vice versa). We proceed greedily, identifying and changing the component of **y** such that the number of unsatisfied parity check equations reduces in each step.

Intuitively, the algorithm in each step attempts to improve the objective function of the minimum-distance decoding, which coincides with the goal of the MEC haplotype assembly. The procedure is terminated when there are no more alleles with a negative drift (with more unsatisfied than satisfied check nodes) or when there are no parity check node violations. The procedure is formalized as Algorithm 1.

Algorithm 1. Bit Flipping Haplotype Assembly

```
1:  procedure BF
2:      $H \leftarrow getParityCheckMatrix$
3:      $\mathbf{c}(0) \leftarrow 1 - 2\tilde{\mathbf{c}}$ set initial values of check nodes to 1 or -1
4:      $t \leftarrow 1$ iterations
5:      repeat
6:          $v_{i,j}^{(t)} \leftarrow c_i(t-1)$ messages from entries to checks
7:          $u_{j,i}^{(t)} \leftarrow \prod_{k \in N_{j}i} v_{k,j}^{(t)}$ messages from checks to entries
8:          $\delta_i(t) \leftarrow -c_i(t-1)(\sum_{j \in N_i} u_{j,i}^{(t)})$ marginal values
9:          if $\min_i(\delta_i(t)) < 0$ then
10:             $k \leftarrow \arg\min(\delta_i(t))$ find the index of highest margin.
11:             $c_k(t) \leftarrow -c_k(t-1)$ flip the value of the entry.
12:             $\forall i \neq k : c_i(t) \leftarrow c_i(t-1)$ keep all other values same
13:         end if
14:         $t \leftarrow t + 1$
15:     until $(t > \mathrm{MAXITER}) \vee (\forall i, j : c_i(t) == u_{j,i}^{(t)}) \vee$
        $(\min_i(\delta_i(t)) \geq 0)$
16: end procedure
```

There are two main sources of randomness in the bit-flipping algorithm—construction of the parity check matrix and breaking ties among flipping variables. For one codebook there are many different code generator matrices with many different orthogonal parity check matrices. We use breadth-first search on the bi-partite graph of the haplotype and read select variables to form a code generator matrix from a random starting point and rely on the Gaussian elimination to obtain the parity check matrix. While performing the bit-flipping algorithm, in case of several variables having the same number of unsatisfied versus satisfied check nodes, we break the ties uniformly at random.

The bit-flipping algorithm was originally proposed for decoding of low-density parity check codes in [22], with the difference that the algorithm there flips more than 1 bit in each iteration. We should also point out that the bit-flipping algorithm is related to the coordinate descent algorithm for l_1-norm minimization, where the goal is to reconstruct a sparse error vector.

4.2 The Belief Propagation Algorithm

Belief propagation is a message-passing scheme for inference in graphical models. A node in the graph receives messages from the nodes its connected to and, based on the received messages, computes and broadcasts its belief about the associated variable. Our belief propagation algorithm works with the code generator matrix and does not require computation of the parity check matrix; for this reason, its computational complexity compares favorably with bit-flipping, as shown in Section 7.

The algorithm takes as input four variables: the starting point sp, the probability of error p_e, the maximal number of iterations MAXITER, and the precision ϵ used as a stopping criterion. In the graphical model representing the haplotype assembly problem, the edges connecting the read nodes with the SNP (haplotype) nodes are associated with the numeric entries in **R**; since those entries are potentially erroneous, we define an auxiliary variable $y_{i,j}$ that accounts for the probability of read errors,

$$y_{i,j} = (1/2 - p_e)(2R(i,j) - 1) + 1/2. \quad (5)$$

In fact, the variables $y_{i,j}$ are the beliefs propagated from the haplotype variables to the read select variables via the XOR function nodes in Fig. 4.

We use bars $^{-}$ to denote the belief complement of a variable; in particular, for any value x, $\bar{x} = 1 - x$. We define the set of active haplotype positions A_H and the set of active reads A_R to be the collection of haplotype and read nodes which received a message on any of their edges in a given iteration, respectively. The belief propagation algorithm for haplotype assembly is formalized as Algorithm 2.

4.3 Complexity

Haplotype assembly concerned with optimizing the MEC criterion is known to be NP-hard [9]. Both bit flipping and belief propagation algorithms are heuristics, and we are interested in characterizing their complexity. Here we provide the complexity analysis in terms of the length of the connected component, i.e., a haplotype block that is connected with the reads. For convenience, we will keep the same notation but with this slightly altered meaning: variable n denotes the length of the connected components of the haplotype (i.e., the length of a haplotype block) and $\mathbf{R}$ is the submatrix of the SNP-fragment matrix containing information relevant to the haplotype block under consideration. We assume that the total number of binary entries in $\mathbf{R}$ is $O(n)$, which is a reasonable assumption given the sparse structure of the SNP-fragment matrix.

Algorithm 2. Belief Propagation Haplotype Assembly

```
1:  procedure BP
2:    $sp \leftarrow randomStartingPoint$
3:    if starting point is a read then
4:      $r_i \leftarrow sp$
5:      $A_R^{(0)} \leftarrow r_i$ (set active reads),
        $A_H^{(0)} \leftarrow \Gamma(A_R^{(0)})$ (set active haplotype)
6:      $b_{r_i}^{(0)} \leftarrow 1 - p_e$ (set initial beliefs),
        $m_{r_i \to h_j}^{(0)} \leftarrow 1 - p_e$ (set initial messages)
7:      $m_{h_j \to r_i}^{(0)} \leftarrow \frac{1}{Z_{h_j}} \prod_{r_k \in A_R^{(0)} \setminus r_i} (m_{r_k \to h_j}^{(0)} y_{k,j} + \bar{m}_{r_k \to h_j}^{(0)} \bar{y}_{k,j})$
        send messages
8:    else
9:      $h_j \leftarrow sp$
10:     $A_H^{(0)} \leftarrow h_j$ set active haplotype
11:     $b_{h_j}^{(0)} \leftarrow 1 - p_e$ (set initial belief), $m_{h_j \to r_i}^{(0)} \leftarrow 1 - p_e$
        (set initial messages)
12:   end if
13:   $t \leftarrow 1$ iterations
14:   repeat
15:     $A_R^{(t)} \leftarrow \Gamma(A_H^{(t-1)})$ update the active reads
16:     $m_{r_i \to h_j}^{(t)} \leftarrow \frac{1}{Zr_{i,j}} \prod_{h_k \in A_H^{(t-1)} \setminus h_j} (m_{h_k \to r_i}^{(t-1)} y_{i,k} + \bar{m}_{h_k \to r_i}^{(t-1)} \bar{y}_{i,k})$
        send messages from $A_R^{(t)}$
17:     $b_{r_i}^{(t)} \leftarrow \frac{1}{Zr_{b_i}} \prod_{h_j \in A_H^{(t-1)}} (m_{h_j \to r_i}^{(t-1)} y_{i,j} + \bar{m}_{h_j \to r_i}^{(t-1)} \bar{y}_{i,j})$
        update read beliefs
18:     $A_H^{(t)} \leftarrow \Gamma(A_R^{(t)})$ update active haplotype
19:     $m_{h_j \to r_i}^{(t)} \leftarrow \frac{1}{Zh_{i,j}} \prod_{r_k \in A_R^{(t)} \setminus r_i} (m_{r_k \to h_j}^{(t)} y_{k,j} + \bar{m}_{(r_k \to h_j)}^{(t)} \bar{y}_{k,j})$
        send messages from $A_H^{(t)}$
20:     $b_{h_j}^{(t)} \leftarrow \frac{1}{Zh_{b_j}} b_{h_j}^{(t-1)} \prod_{r_i \in A_R^{(t)}} (m_{r_i \to h_j}^{(t)} y_{i,j} + \bar{m}_{r_i \to h_j}^{(t)} \bar{y}_{i,j})$
        update haplotype beliefs
21:     $t \leftarrow t + 1$
22:   until $(t >$ MAXITER$) \vee (\|\mathbf{b}_{\mathbf{h}}^{(t)} - \mathbf{b}_{\mathbf{h}}^{(t-1)}\|_2 \leq \epsilon)$
23: end procedure
```

We distinguish between time and space complexity. Time complexity characterizes the minimum time needed to obtain the output of the algorithm and is concerned with its longest sequential path. This measure is particularly interesting when running algorithms on distributed systems. On the other hand, space complexity characterizes the algorithm's memory requirements.

The bit-flipping algorithm. The time complexity for the bit-flipping algorithm is $O(n)$. To show this, we divide the procedure in two major steps: finding the parity check matrix and running the bit flipping algorithm. The time complexity of finding the parity check matrix from the code generator matrix is $O(n)$. This is lower than the standard Gaussian elimination since we arrange the code generator matrix to have lower triangular form by following an $O(n)$ breadth-first search that permutes the haploype and read select variables in an appropriate fashion. In addition, each pivot has only one non-zero value above it, and thus the time needed for the transformation to the fully reduced form is again $O(n)$. On the other hand, the maximal number of iterations of the bit flipping algorithm is $O(n)$. The number of iterations is upper bounded by the number of unsatisfied parity check nodes. Since at each iteration this number is reduced by at least 1, the maximal number of iterations is upper bounded by the number of check nodes in the parity check matrix. Now, from the procedure used to generate parity check matrix one can observe that the number of check nodes in the parity check matrix is strictly less than the number of entries in $\mathbf{R}$, which is again given by $O(n)$. The space complexity of the BF algorithm does not exceed $O(n^2)$ since the number of connections between parity check nodes and observed alleles is at most quadratic in n.

The belief propagation algorithm. The time complexity of the belief propagation algorithm is bounded by the number of iterations. Unlike the bit-flipping algorithm, the belief propagation algorithm does not need the transformation from the code generator matrix to the parity check matrix. The number of iterations of the belief propagation algorithm is upper bounded by the MAXITER variable. Note that MAXITER variable needs to be larger than the depth of the graphical model since we want all the SNPs and read select variables to be in the active set. Also note that in each iteration all the values of the active nodes need to be updated, whereas in the bit flipping algorithm only a single variable is updated. The space complexity of the belief propagation algorithm is $O(n)$ since all that is required to store is the description of the original graphical model.

Initialization and reruns (for both algorithms). Performance of both algorithms depends on the starting point; hence to obtain better results we repeat the execution of the algorithm from different starting points. We use a random restart and keep searching for the solution as long as the MEC score improves. The number of times we rerun the algorithms is adaptively determined based on the incremental change in the MEC score.

5 Limits of Performance of Haplotype Assembly

We find analytical expressions for the achievable accuracy of the haplotype assembly problem concerned with optimization (4). In particular, we compute the lower bounds on the probability of haplotype assembly and switch errors, and the expected number of SNP and switch errors.

To start, in Fig. 5 we introduce a simple bipartite graph model of the SNP fragment matrix. The nodes on the left correspond to the haplotype/SNP values (i.e., components of $\mathbf{h}$) while the nodes on the right correspond to the read select variables (components of $\mathbf{s}$). The edge between a SNP and a read select node exists if the read covers the SNP and provides information about its allele. Here the actual values of the observed alleles are not important since the error probability characterizing the performance of a linear block code is the property of a codebook, not a specific codeword. In Fig. 5 there are three examples of cuts that will be of interest in our analysis. Cut C1 isolates a single SNP node, and we define the cardinality of that cut to be equal to the

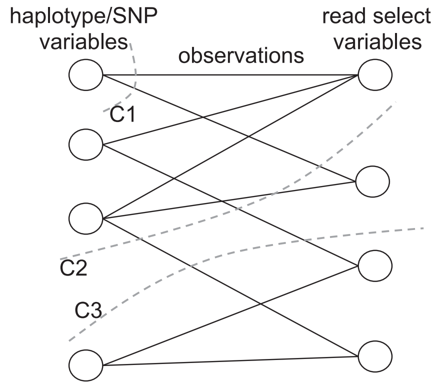

Fig. 5. A sample bipartite graph representation of the fragment matrix. The nodes on the left represent SNPs while the nodes on the right represent read select variables. The edge between a SNP and a read select node exists if the read covers the SNP location. Three cuts are illustrated in the figure: C1, C2, and C3. We refer to the cut C1 as isolating cut since it contains all the edges emanating from a single SNP, and define the cardinality of the cut as the coverage of the corresponding SNP. We refer to the cuts C2 and C3 as separating cuts since they partition SNPs into two non empty sets.

coverage of the isolated SNP. Cuts C2 and C3 partition the SNP nodes in two non-empty sets.

The probability of error, defined as the probability that the assembled haplotype is different from the true one, is given by

$$P(\hat{\mathbf{h}} \neq \mathbf{h}) \geq 1/2 \sum_{i=\lfloor k/2 \rfloor+1}^{k} \binom{k}{i} p^i (1-p)^{k-i},$$

where p denotes the SNP calling error rate and k is the smallest coverage over all SNP positions (the SNP with the lowest coverage is the most error prone). If the number of SNP calling errors is large ($> \lfloor k/2 \rfloor + 1$), it is likely that the algorithms will arrive at a haplotype sequence with an erroneous SNP position. In the bipartite graph representation in Fig. 5, this bound corresponds to the errors introduces along the smallest isolating cut.

Accuracy of haplotype assembly is often expressed in terms of the switch probability. Switch refers to the event where a subsequence of consecutive errors starts at a site in the assembled haplotype; the errors essentially imply mistaking a segment of $\hat{\mathbf{h}}$ for that of of $\mathbf{h}$. In the bipartite graph representation each separating cut potentially leads to the switch in the decoded haplotype. Now, looking at all possible locations of switches, it is straightforward to show that the probability of a switch can be bounded by

$$P_{\text{switch}} \geq 1/2 \sum_{i=\lfloor s/2 \rfloor+1}^{s} \binom{s}{i} p^i (1-p)^{s-i},$$

where s denotes the minimum separating cut of the bipartite graph representation of fragment matrix. Notice that the switch probability is always greater than the probability of an erroneous SNP, as the smallest isolating cut is just one possible separating cut in the graph.

Next, we examine the expected number of erroneous (i.e., inverted) SNPs. This expectation is lower bounded by the coverage errors as

$$\begin{aligned} E[\|\hat{\mathbf{h}} - \mathbf{h}\|_0] &= E\left[\sum_{i=1}^{n} \mathbf{1}_{\hat{h}_i \neq h_i}\right] \\ &= \sum_{i=1}^{n} P(\hat{h}_i \neq h_i) \\ &\geq \sum_{i=1}^{n} \frac{1}{2} \sum_{j=\lfloor \frac{k_i}{2} \rfloor+1}^{k_i} \binom{k_i}{j} p^j (1-p)^{k_i-j}, \end{aligned} \tag{6}$$

where k_i denotes the coverage of the ith SNP. Note that on the right-hand side in (6) we sum up the probabilities of each SNP being inverted due to the errors introduced into the corresponding isolating cut. This leads to a lower bound because all isolating cuts in the bipartite graph are independent, i.e., each edge of the bipartite graph appears in at most one isolating cut.

For arbitrary separating cuts in the graph, the independence assumption does not hold (for example, cuts C2 and C3 are not independent since they share two edges). We need to take that into account when characterizing the lower bound on the expected number of switches,

$$\begin{aligned} E[\text{SWER}] &= \frac{1}{n} \sum_{i=2}^{n} P(\text{switch between } i-1 \text{ and } i) \\ &\geq \frac{1}{n} \sum_{i=2}^{n} \frac{1}{2} \sum_{j=\lfloor \frac{s_i}{2} \rfloor+1}^{s_i} \binom{s_i}{j} p^j (1-p)^{s_i-j}. \end{aligned}$$

Here s_i denotes the smallest cut separating the ith SNP node that is independent of all other separating cuts $s_j \neq s_i$. It readily follows that

$$E[\text{SWER}] \geq \sum_{i=1}^{n} \frac{1}{2} \sum_{j=\lfloor \frac{k_i}{2} \rfloor+1}^{k_i} \binom{k_i}{j} p^j (1-p)^{k_i-j},$$

where k_i denotes the coverage of the ith SNP.

We should point out that the lower bound on the probability of switching is always greater than the lower bound on the probability of error under maximum a posteriori decoding. For the scenario where the data is characterized by long reads and low coverage (as in, e.g., Fosmid datasets), the computed bounds are fairly tight; however, when the coverage is high and the reads are relatively short (resembling reads in *1,000 Genomes Project* datasets), they tend to be loose. Either way, they provide a useful insight about the achievable accuracy of single individual haplotyping (SIH).

6 Polyploid Haplotype Assembly

In previous sections, we focused on haplotype assembly for diploids. We now turn our attention to the polyploid version of the haplotype assembly problem. For simplicity of presentation, we constrain SNPs to be bi-allelic; extension to the multi-allelic scenario is relatively straightforward (albeit somewhat cumbersome) and involves replacing vectors of beliefs by matrices. Hence, the alphabet used to denote alleles in the SNP data matrix remains $\{0, 1, \times\}$, where $\times$ marks the positions in a read not covering the corresponding SNP sites.

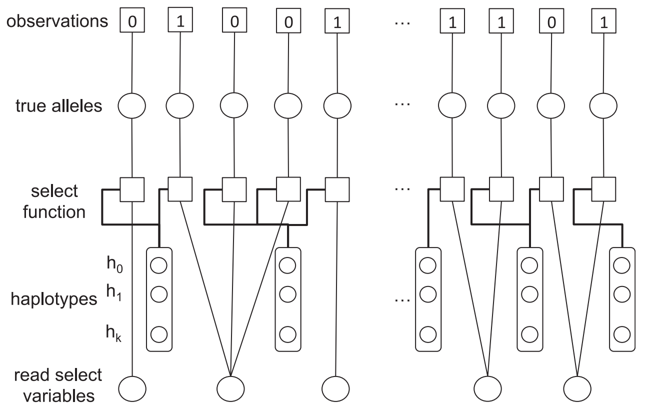

Fig. 6. An illustration of the graphical model used for polyploid haplotype assembly. This model is an extension of the model in Fig. 3. The modified belief propagation algorithm is implemented on this graph.

The goal of polyploid haplotype assembly concerned with optimizing the MEC criterion[1] is to minimize Z over the set $(\mathbf{h}_0, \mathbf{h}_1, \ldots, \mathbf{h}_{k-1})$,

$$Z = \sum_{i=1}^{m} \min(\mathrm{hd}(\mathbf{r}_i, \mathbf{h}_0), \mathrm{hd}(\mathbf{r}_i, \mathbf{h}_1), \ldots, \mathrm{hd}(\mathbf{r}_i, \mathbf{h}_{k-1})), \qquad (7)$$

where k denotes the ploidy and $\mathrm{hd}(\cdot,\cdot)$ is the generalized Hamming distance between its arguments.

The above minimization can be rewritten using a select variable vector $\mathbf{s}$. The definition of the select variables is somewhat modified in the polyploid case: the ith component of $\mathbf{s}$, s_i, is the index of the haplotype that the read is nearest to in terms of the generalized Hamming distance. This means that the optimization of the MEC objective can be rewritten as

$$\min_{\mathbf{s}\in\{0,\ldots,k-1\}^m} \sum_{i=1}^{m} \sum_{j=0}^{k-1} \mathbf{1}_{\{s_i==j\}} \mathrm{hd}(\mathbf{r}_i, \mathbf{h}_j). \qquad (8)$$

The graphical model that describes the data used for polyploid assembly is illustrated in Fig. 6. Since the read select variables (i.e., components of $\mathbf{s}$) no longer take the binary values $\{0, 1\}$, the linear coding analogy does not extend to this scenario. However, relying on the graph in Fig. 6, we can still design a belief propagation algorithm for polyploid haplotype assembly. Note that the select function nodes of the graph shown in Fig. 6 act as a simple multiplexer, allowing only one of the entries $\{\mathbf{h}_0, \mathbf{h}_1, \ldots, \mathbf{h}_k\}$ to "pass" depending on the value of the select variable $\mathbf{s}$.

6.1 The Polyploid Belief Propagation Algorithm

To extend the belief propagation algorithm from Section 4 to the assembly of haplotypes in polyploids, we need to redefine the nodes that participate in the exchange of the beliefs as well as specify messages that are being exchanged. For each SNP location, we need to infer the probabilities of the alleles for each of the k haplotypes. To this end, for the ith component of $\mathbf{h}$ we define a variable $p_{\mathbf{h}}^{(i)}$ as the probability vector with k entries. The value of the jth entry in this vector, $p_{\mathbf{h},j}^{(i)}$, is the probability of the jth haplotype having a reference allele at the ith position. Similarly, for the ith component of $\mathbf{s}$ we define a probability vector $p_{\mathbf{s}}^{(i)}$ of length k, where the jth component of $p_{\mathbf{s}}^{(i)}$, $p_{\mathbf{s},j}^{(i)}$, is the beliefs that the ith read is associated with the jth haplotype ($1 \leq j \leq k$).

Note a major distinction between the probability vectors $p_{\mathbf{h}}^{(i)}$ and $p_{\mathbf{s}}^{(i)}$. On one hand, an allele may occur at a particular position in multiple haplotype strands, and the information about an allele in one haplotype may not help infer alleles at the same position in other haplotypes. On the other hand, each read should eventually be associated with a single haplotype strand, thus helping uniquely determine multiple SNP positions in the haplotype. This difference between the probability vectors motivates different ways of aggregating messages at the graph nodes. Since SNP variables act as factors in the belief propagation factor graphs, imposing the presence of an allele in a haplotype, messages at the corresponding nodes are computed via sum aggregation. The read select variables are the ones for which we want to find the most likely configuration via marginalization, leading us to the product form.

The auxiliary variable $y_{i,j}$ has the same definition as in the belief propagation algorithm for diploids (see Eq. (5)). However, we modify the notation $^{-}$ to denote the belief complement of a vector variable; in particular, for any probability vector $\mathbf{x}$ of size k, $\bar{\mathbf{x}} = (1-\mathbf{x})/(k-1)$.

The starting point of the algorithm is restricted to a read, and a separate starting point is chosen for each strand of the haplotype. A different read will be uniquely associated with each one of the haplotype strands, as seen in Step 22 of the polyploid belief propagation algorithm.

The algorithm is formalized as Algorithm 3.

7 Simulation Results and Discussion

We test the performance of the proposed algorithms on both real and simulated data sets. The experimental data include the *1,000 Genomes Project* [29] and *Fosmid* [27] data sets, on which we compare the MEC scores and runtimes of bit-flipping and belief propagation algorithms with those of HapCUT [19], HapCompass [18] and RefHap [30]. RefHap is part of the Single Individual Haplotyping package that includes DGS [17], FastHare [31], SHRThree [32], Speedhap [33], 2d-MEC [34], and WMLF [35]. These algorithms were compared against each other as well as against HapCUT [19] in [36]. We also tested all of the aforementioned algorithms from the SIH package on the Fosmid dataset and found that RefHap outperforms others in terms of both speed and accuracy. On another note, simulated data mimics long (fosmid-like) pair-end reads with varying coverages, allowing comprehensive study of the algorithms' performance in terms of switch error rates which we can compute due to availability of the ground truth. The algorithms were run on MacBook Air with 2.13 GHz Intel Core Duo processor with 4 GB of DDR3 RAM and MacBook with

1. It was remarked in [28] that the MEC objective may not be suited for use in the polyploid case due to ambiguity when phasing with reads that cover the same collection of SNP sites. However, when the reads are long and cover diverse subsets of SNP sites, as is the case with recent high-throughput technologies such as fosmid, the MEC objective facilitates successful assembly as indicated by the results presented in Section 7.

2.4 GHz Intel Core i5 processor with 4 GB of DDR3 RAM. Implementation of the algorithms is available for download from http://users.ece.utexas.edu/~hvikalo/DecodingBFBP.html and http://sourceforge.net/projects/bfbp.

Algorithm 3. Polyploid Belief Propagation Haplotype Assembly

```
1:  procedure PBP
2:      $sp \leftarrow randomStartingPoint$
3:      $order \leftarrow 1$
4:      $r_i \leftarrow sp$
5:      $A_R^{(0)} \leftarrow r_i$ (set active reads)
6:      $b_{r_i}^{(0)} \leftarrow [1-p_e \;\; \frac{p_e}{k-1} \;\; \dots \;\; \frac{p_e}{k-1}]$ (set initial beliefs),
        $m_{r_i \to h_j}^{(0)} \leftarrow [1-p_e \;\; \frac{p_e}{k-1} \;\; \dots \;\; \frac{p_e}{k-1}]$ (set initial messages)
7:      $t \leftarrow 1$ iterations
8:      repeat
9:          $A_H^{(t)} \leftarrow \Gamma(A_R^{(t-1)})$ update active haplotype
10:         if $order = k$ then
11:             $bias_j \leftarrow$ vector with $h_j$ degree on the reference number
                of largest beliefs
12:         else
13:             $bias_j \leftarrow [0 \;\; 0 \;\; \dots \;\; 0]$
14:         end if
15:         $m_{h_j \to r_i}^{(t)} \leftarrow \frac{1}{Zh_{i,j}}[bias_j + \sum_{r_l \in A_R^{(t)} \setminus r_i} (m_{r_l \to h_j}^{(t)} y_{l,j} + \bar{m}_{(r_l \to h_j)}^{(t)} \bar{y}_{l,j})]$
            send messages from $A_H^{(t)}$
16:         $b_{h_j}^{(t)} \leftarrow \frac{1}{Zh_{b_j}} b_{h_j}^{(t-1)} [bias_j + \sum_{r_i \in A_R^{(t)}} (m_{r_i \to h_j}^{(t)} y_{i,j} + \bar{m}_{r_i \to h_j}^{(t)} \bar{y}_{i,j})]$
            update haplotype beliefs
17:         $A_R^{(t)} \leftarrow \Gamma(A_H^{(t-1)})$ update the active reads
18:         $b_{r_i}^{(t)} \leftarrow \frac{1}{Zr_{b_i}} \prod_{h_j \in A_H^{(t-1)}} (m_{h_j \to r_i}^{(t-1)} y_{i,j} + \bar{m}_{h_j \to r_i}^{(t-1)} \bar{y}_{i,j})$
            update read beliefs
19:         $m_{r_i \to h_j}^{(t)} \leftarrow \frac{1}{Zr_{i,j}} \prod_{h_l \in A_H^{(t-1)} \setminus h_j} (m_{h_l \to r_i}^{(t-1)} y_{i,l} + \bar{m}_{h_l \to r_i}^{(t-1)} \bar{y}_{i,l})$
            send messages from $A_R^{(t)}$
20:         if $order < k$ then
21:             $r_c \leftarrow$ with lowest margin of belief that it belongs to
                haplotypes 1 to $order$
22:             $b_{r_c}^{(t)} \leftarrow [\frac{p_e}{k-1} \;\; \dots \;\; 1-pe \;\; \dots \;\; \frac{p_e}{k-1}]$ assign a read to
                haplotype $h_{order}$
23:             $m_{r_c \to h_j}^{(t)} \leftarrow [\frac{p_e}{k-1} \;\; \dots \;\; 1-pe \;\; \dots \;\; \frac{p_e}{k-1}]$ update the
                messages for this read
24:             $order \leftarrow order + 1$
25:         end if
26:         $t \leftarrow t + 1$
27:     until $(t > \text{MAXITER}) \vee (\|\mathbf{b}_\mathbf{h}^{(t)} - \mathbf{b}_\mathbf{h}^{(t-1)}\|_2 \leq \epsilon)$
28: end procedure
```

7.1 Benchmarking on 1,000 Genomes Project Data

For the first test of the proposed algorithms we relied on a data set from the *1,000 Genomes Project*—in particular, the sample NA12787 sequenced at high coverage using 454 sequencing platform, also considered in several recent studies including [18]. We compared the MEC scores achieved by our algorithms to those of the widely used HapCUT [19] and the more recent RefHap [30] and HapCompass [37]. HapCompass, RefHap and HapCUT were called with their default settings; for HapCUT that means using software version v.05 and running standard 100 iterations, for RefHap it means version of SIH package 1.0, while for HapCompass we ran software version 0.7.1. HapCompass processed BAM and VCF files downloaded from the 1,000 Genomes Project data repository and its resulting fragment matrix was partitioned into connected components. Each component was separately fed into HapCUT along with the corresponding quality scores, ensuring that both algorithms take into account the same SNP locations and provide a fair comparison of the MEC scores (i.e., the number of SNPs, reads, alleles and blocks was the same for all the algorithms). The resulting MEC scores are reported in the second super-column of Table 2. As can be seen there, among all algorithms belief propagation provides the smallest MEC score for all chromosomes. Bit-flipping (BF) is slightly worse than belief propagation, matching its performance on four chromosomes, but better than HapCUT on all and better than HapCompass and RefHap on all but one chromosome. Note that RefHap does not phase all the SNPs and discards a fraction of reads which leads to an unrealistic MEC score that is artificially lower than those achieved by the other algorithms. To make a fair comparison, we assigned the reads discarded by RefHap to either $\mathbf{h}$ or $\bar{\mathbf{h}}$ using the phased SNPs and then phased the remaining SNPs in such a way that the resulting total MEC score of RefHap is the lowest possible. The computational overhead due to these additional operations is not included in the reported RefHap runtimes; for completeness, the original RefHap MEC scores evaluated on a reduced set of reads are reported in parenthesis.

When comparing the speed of the algorithms, we wanted to exclude the bias due to the system calls for I/O operations; therefore, in the table we report the user time provided by the UNIX time command. For HapCUT, the total time including the system calls, operations and context switches is around an order of magnitude greater than the times reported in Table 2. For HapCompass, the total time in only marginally greater than the user time given in the table. The BP and BF algorithms also have the total time only marginally greater than the user time. As evident from the table, RefHap is the fastest among the considered schemes but its speed comes at the cost of reduced accuracy. On the other hand, the BP and BF algorithms are faster than HapCUT and HapCompass except for chromosome 6 where HapCUT incurs the least amount of runtime.

7.2 Fosmid Data

Fosmid pool-based sequencing provides very long fragments, characterized by much higher ratio of the number of SNPs to the number of reads than in standard high-throughput sequencing platforms. We consider the fosmid sequence data for a HapMap NA12878, also studied in [27]. As an example, chromosome 1 of this dataset has 22,737 reads and 122,960 SNPs.

We compared the performance of our bit-flipping and belief propagation algorithms to that of HapCUT and RefHap and report the results in Table 3 (we attempted running HapCompass on the same dataset but that algorithm was running out of memory or stopping for unknown reasons). As can be seen from Table 3, BF or BP outperform HapCUT on half of the chromosomes; the MEC scores of all three algorithms are within 1 percent margin for all

TABLE 2
MEC Scores and Execution Times for HapCUT (HCUT), HapCompass (HCom), Bit-Flipping, and Belief Propagation Algorithms for the *1,000 Genomes Project* Individual NA12787

	Properties				MEC score					Execution time (s)				
Chr.	#SNP	#Read	#Allele	#blocks	HCUT	HCom.	RefHap	BF	BP	HCUT	HCom.	RefHap	BF	BP
1	122,960	180,199	403,138	21,825	12,675	12,390	12,380 (11,203)	12,312	**12,310**	115	214	28	59	39
2	139,475	211,311	475,770	24,836	15,363	15,015	14,991 (13,650)	14,933	**14,916**	134	243	61	117	99
3	117,657	180,572	407,084	20,855	13,243	12,970	12,947 (11,777)	12,882	**12,872**	113	203	37	70	60
4	119,330	190,029	437,636	20,802	14,980	14,642	14,631 (13,361)	14,536	**14,532**	123	202	45	116	89
5	112,643	171,881	387,704	20,049	12,611	12,266	12,260 (11,070)	12,200	**12,196**	122	167	30	62	48
6	116,414	189,932	463,272	19,579	17,057	16,805	16,800 (15,474)	16,924	**16,769**	202	16,290	987	1,193	1,073
7	94,511	148,305	340,748	16,624	11,445	11,174	11,165 (10,098)	11,110	**11,108**	97	164	26	96	51
8	94,024	152,864	349,66	16,571	11,088	10,817	10,797 (9,750)	10,735	**10,732**	97	133	21	60	60
9	71,898	115,722	2634,19	12,979	8,481	8,319	8,279 (7,503)	8,237	**8,236**	72	149	32	140	43
10	85,499	136,288	310,879	15,001	10,167	9,899	9,893 (9,030)	9,833	**9,828**	86	122	21	61	47
11	81,018	126,027	288,307	14,225	9,308	9,104	9111 (8,273)	9,047	**9,042**	83	131	25	64	68
12	78,146	117,673	265,958	13,849	8,659	8,418	8,4,00 (7,651)	8,362	**8,361**	76	131	17	40	32
13	63,689	100,081	230,321	11,241	7,848	7,695	7,687 (7,041)	7,652	**7,631**	64	108	40	80	67
14	53,934	82,139	185,435	9,598	5,866	5,715	5,698 (5,154)	5,662	**5,661**	51	84	11	33	29
15	46,254	73,559	166,860	8,191	5,244	5,137	5,120 (4,660)	5,102	**5,100**	45	77	9	25	18
16	51,786	86,684	201,865	9,043	6,400	6,231	6,205 (5,708)	6,177	**6,175**	54	83	15	36	51
17	38,839	58,363	134,103	6,696	4,631	4,550	4,537 (4,201)	**4,506**	**4,506**	37	84	17	69	20
18	49,873	77,291	175,107	8,821	5,531	5,371	5,363 (4,868)	5,342	**5,340**	47	75	8	34	23
19	31,760	46,397	105,525	5,602	3,571	3,421	3,403 (3,086)	**3,392**	**3,392**	29	54	10	15	11
20	38,044	58,879	134,089	6,831	4,213	4,122	4,103 (3,754)	4,087	**4,086**	38	61	7	19	14
21	24,342	40,107	92,582	4,379	3,021	2,987	2,972 (2,695)	**2,953**	**2,953**	26	40	6	14	13
22	22,801	33,671	77,122	4,076	2,395	2,324	2,316 (2,112)	**2,301**	**2,301**	24	37	5	11	7

chromosomes. At the same time, both bit-flipping and belief propagation are significantly faster than HapCUT—the widest gap in speed is seen on chromosome 6 where belief propagation is about 40 times faster than HapCUT. RefHap is again the fastest among the considered schemes but its speed is traded off for accuracy. As before, the reported run-times exclude the system calls time.

7.3 Simulation Results: The Diploid Case

We simulate two scenarios, one with high coverage paired-end reads that resemble those in *1,000 Genomes Project* data-sets and the other with long reads and low coverage similar to what may be available in a *Fosmid* dataset.

To emulate the short-read high-coverage scenario, we generated reads that span 500 basis with inserts having 10 k

TABLE 3
MEC Scores and Time of Execution for HapCUT, Bit-Flipping, and Belief Propagation Algorithms on the Fosmid Dataset for NA12878 Individual

	Properties				MEC score				Execution time (s)			
Chr.	#SNP	#Read	#Allele	#blocks	HapCUT	RefHap	BF	BP	HapCUT	RefHap	BF	BP
1	122,960	22,736	393,201	1,316	9,555	9,644 (8,051)	9,589	**9,552**	2,423	2.3	319	268
2	129,732	22,602	413,205	1,519	9,668	9,728 (7,910)	9,734	**9,661**	3,089	2.22	254	282
3	108,204	18,722	330,763	1,285	7,566	7,635 (6,111)	7,606	**7,554**	2,124	2.01	220	213
4	107,430	16,027	306,639	1,372	6,267	6,317 (4,880)	**6,262**	6,267	2,431	1.81	159	234
5	103,442	17,013	317,206	1,196	**6,922**	6,963 (5,558)	6,960	6,946	2,202	1.91	238	213
6	107,882	16,451	332,388	1,072	**7,957**	8,038 (6,341)	8,002	7,965	11,260	2.02	286	283
7	87,563	14,936	275,308	1,011	**6,071**	6,098 (4,961)	6,107	6,078	2,242	1.79	274	208
8	86,708	13,733	275,302	959	6,260	6,289 (5,092)	6,282	**6,250**	2,670	1.78	332	158
9	66,996	11,528	224,982	678	5,464	5,505 (4,591)	**5,460**	5,462	2,309	1.56	286	163
10	79,978	14,064	265,142	779	**6,446**	6,489 (5,357)	6,475	6,475	2,006	1.62	180	199
11	75,235	13,519	246,499	802	**5,560**	5,602 (4,620)	5,575	5,567	1,752	1.64	177	182
12	72,917	13,377	238,070	793	**5,666**	5,691 (4,686)	5,692	5,670	1,610	1.71	183	189
13	57,287	8,800	168,625	671	3,968	4,030 (3,155)	**3,961**	3,967	1,276	1.28	108	92
14	50,219	9,030	165,775	523	**3,979**	4,017 (3,244)	4,016	3,989	1,302	1.26	168	186
15	43,578	8,306	149,536	481	4,009	4,053 (3,341)	4,030	**4,003**	1,083	1.17	120	150
16	49,736	9,655	191,480	400	**5,087**	5,102 (4,438)	5,128	5,099	2,481	1.58	220	260
17	37,820	8,776	146,019	426	4,744	4,819 (4,159)	4,773	**4,740**	1,069	1.21	151	191
18	46,313	7,704	146,353	497	3,448	3,473 (2,801)	3,475	**3,441**	1,021	1.17	83	88
19	30,777	7,431	119,629	266	**3,900**	3,940 (3,406)	4,037	3,902	773	1.07	137	136
20	36,398	7,447	135,745	317	3,811	3,863 (3,295)	3,883	**3,810**	951	1.2	164	115
21	22,756	3,760	73,711	222	**1,953**	1,967 (1,601)	1,958	**1,953**	587	0.72	48	33
22	22,083	5,567	92,889	141	**3,261**	3,343 (2,876)	3,375	3,278	752	1.24	218	188

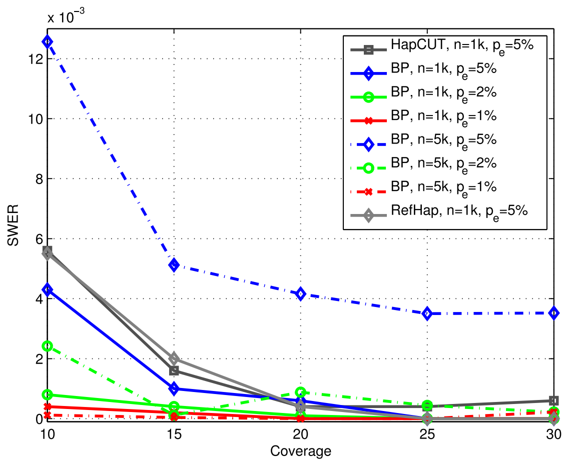

Fig. 7. Simulated SWER rates for haplotype assembly with short reads using the BP algorithm (diploid). Results are averaged over 10 different fragment matrices, and reported with respect to varying coverage, block length, and probability of error.

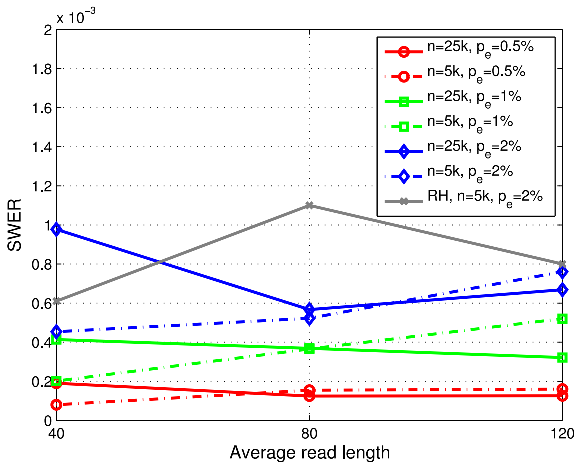

Fig. 8. Simulated SWER rates for haplotype assembly with long reads using the BP algorithm (diploid). Results are averaged over 10 different fragment matrices, and reported with respect to varying read length, block length, and probability of error.

mean length and 10 percent deviation. The rate of SNPs is assumed to be 1 in 300 basis as reported in [38]. We simulate sampling of the entire genome with the paired-end reads, marking each base as a SNP location with probability 1 in 300. This means that the number of basis between two neighboring SNP locations is a geometrically distributed random variable (as assumed in, e.g., [28]).

We study the dependence of the switch error rate on coverage. Moreover, we are interested in understanding how haplotype assembly depends upon the errors in the SNP fragment matrix. To this end, we consider haplotype block lengths of 1,000 and 5,000, with the coverage of 10, 15, 20, 25 and 30. Fore each pair of parameters (block length and coverage), we simulate error rates of 1, 2 and 5 percent in the SNP fragment matrix. The error rate of 2 percent is the closest to what we observed in experimental data sets (both 1,000 Genomes Project and Fosmid). Each experiment is repeated 10 times. The mean values of SWER for the BP algorithm are reported in Fig. 7. As can be seen from the figure, increase in the block length leads to the increase in the SWER value for the same value of the erroneous data rate and coverage. Increasing the fraction of errors in the SNP fragment matrix causes deterioration of the SWER performance, while increasing the coverage improves the SWER. Note that even in the worst considered scenario (highest rate of sequencing errors, largest block size, and smallest coverage), SWER remains below 2 percent. For a comparison, we include the SWER of RefHap and HapCUT for block lengths $n = 1{,}000$ and error rate 5 percent. Except for the coverage $c = 20$, the BP algorithms leads to lowest SWERs for the considered set of parameters.

Next, we consider the long reads, small coverage scenario (similar to the Fosmid data). The reads are generated with lengths distributed following Poisson distribution with specified average read length. Within each read, a single SNP is covered independently with probability 0.9. We study the performance of our algorithm in terms of SWER. The parameters of the simulation are the number of SNPs, the sequencing error rate, and the average read length. The number of SNPs (i.e., haplotype block length) is 5,000 and 25,000. The error rate in the SNP fragment matrix is set to 0.5, 1 and 2 percent. The average read length was set to 40, 80, and 120. The coverage was set to 8. Each experiment was repeated 10 times and the mean value and standard deviation of SWER are shown in the Fig. 8. We again include the comparison with RefHap for block lengths $n = 5{,}000$ and error rate 2 percent. As can be seen from the figure, the BP algorithms provides better SWERs for all coverage levels.

We see from Fig. 8 that, as expected, higher data error rates lead to higher SWER. Interestingly, the blocks of length 5,000 seem to have similar SWER to those for the blocks of length 25,000. Moreover, long reads appear to lead to SWER similar to those provided by short reads. In order to gain further insight, we examined the cause of the switches. As it turns out, majority of the switches are actually single SNPs that got inverted, implying that most of the reported errors in haplotype assembly are due to errors in the isolated cuts (discussed in Section 5).

7.4 Simulation Results: The Polyploid Case

We implemented our belief propagation algorithm for haplotype assembly of polyploids in C, and compared its performance in terms of MEC, SWER and execution time to the polyploid version of the HapCompass algorithm [18]. [We attempted to compare its performance to HapTree [28] as well, however, for the considered block sizes HapTree runs out of memory.] We simulated pair-end reads with the same setup as in the diploid simulations, for haplotype block lengths 200 and 1,000. The coverage was varied and the error rate in the SNP fragment matrix was 1 and 2 percent. The plotted lines are obtained by averaging results of 10 simulation runs.

The MEC scores obtained by applying the algorithms to the assembly of a triploid are given in Fig. 9. As can be seen there, the BP algorithm achieves significantly lower MEC scores than HapCompass. As the data error rates increase, the MEC scores increase by approximately the same factor.

Next, we study the switch error rate and compare the performance of the algorithms for various polyploid orders. The results are shown in Fig. 10. As we can see from the figure, increasing the coverage reduces the SWER while as the ploidy increases the SWER deteriorates.

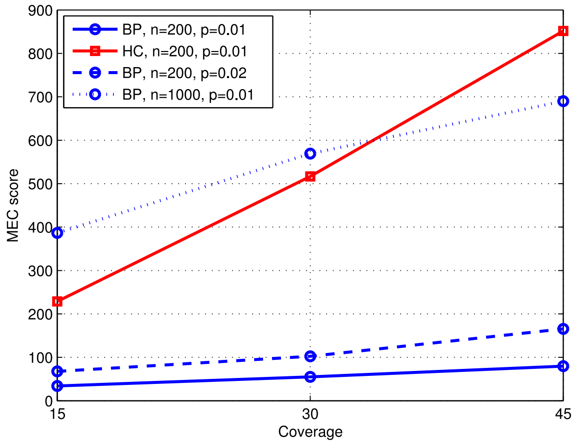

Fig. 9. Simulated MEC scores for haplotype assembly of a triploid (the belief propagation algorithm and HapCompass).

Finally, we study running times of the belief propagation algorithm for haplotype assembly of polyploids and report them in Fig. 11. As can be seen there, the run time increases with coverage, ploidy and block size, while it appears independent of data error rates. Moreover, the belief propagation algorithm is significantly faster than HapCompass for the same sets of simulation parameters. Note that the reported run times are obtained using the UNIX time command (i.e., we are reporting the user time).

8 Conclusion and Future Work

We proposed and studied formulation of the haplotype assembly problem that relies on a novel graphical representation and draws upon parallels between haplotype assembly and decoding in data communication systems. We proposed two algorithms, namely the bit-flipping and belief propagation algorithm, both highly accurate and fast heuristics. The complexity of both algorithms is only linear in the length of the haplotype block. Their accuracy compares favorably with HapCUT, HapCompass and RefHap on both simulated and experimental data. When applied to fosmid data characterized by long fragments and small ratio between the number of reads and haplotype length, our proposed methods are often more than 10 times faster than HapCUT. Moreover, we extended the belief propagation algorithm to the haplotype assembly of polyploids, focusing on the bi-allelic case, and demonstrated significant performance improvements over HapCompass.

As part of the future work, it is of interest to explore other, more sophisticated, decoding algorithms in the context of haplotype assembly (e.g., belief propagation with soft thresholding). There is also a number of potentially interesting fundamental questions such as performance versus complexity tradeoffs and further analysis of the achievable limits of performance.

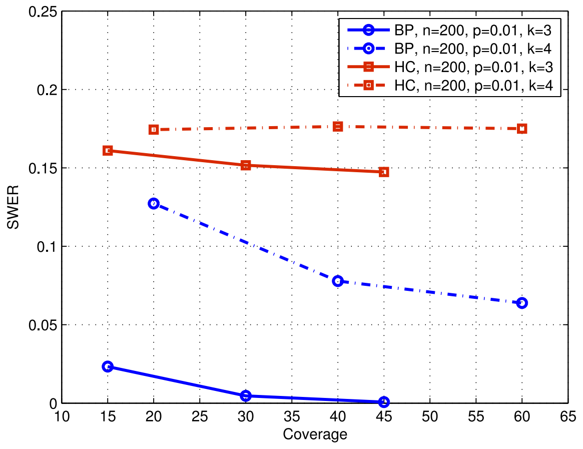

Fig. 10. Simulated SWER rates for haplotype assembly of various polyploids (the belief propagation algorithm and HapCompass). Haplotype block length is set to 200.

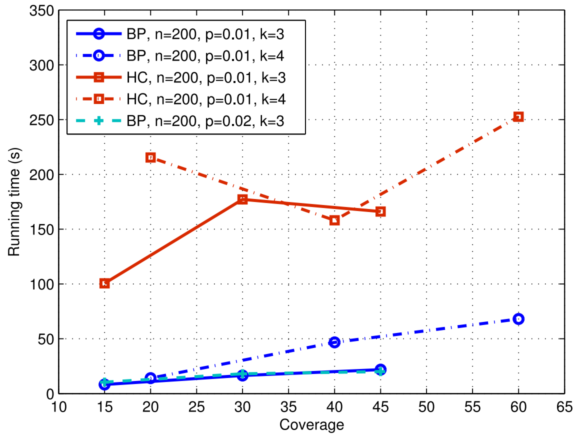

Fig. 11. Comparison of running times for haplotype assembly of various polyploids (the belief propagation algorithm and HapCompass). Haplotype block length is set to 200.

Acknowledgments

This work was supported in part by the US National Science Foundation under grant CCF-1320273.

References

[1] M. Ronaghi, S. Karamohamed, B. Pettersson, M. Uhlén, and P. Nyrén, "Real-time dna sequencing using detection of pyrophosphate release," *Analytical Biochemistry*, vol. 242, no. 1, pp. 84–89, 1996.
[2] N. Hall, "Advanced sequencing technologies and their wider impact in microbiology," *J. Exp. Biol.*, vol. 210, no. 9, pp. 1518–1525, 2007.
[3] E. R. Mardis, "The impact of next-generation sequencing technology on genetics," *Trends Genetics*, vol. 24, no. 3, pp. 133–141, 2008.
[4] S. C. Schuster, "Next-generation sequencing transforms todays biology," *Nature*, vol. 200, no. 8, pp. 16–18, 2007.
[5] E. S. Lander, L. M. Linton, B. Birren, C. Nusbaum, M. C. Zody, J. Baldwin, K. Devon, K. Dewar, M. Doyle, W. FitzHugh, et al., "Initial sequencing and analysis of the human genome," *Nature*, vol. 409, no. 6822, pp. 860–921, 2001.
[6] J. C. Venter, M. D. Adams, E. W. Myers, P. W. Li, R. J. Mural, G. G. Sutton, H. O. Smith, M. Yandell, C. A. Evans, R. A. Holt, et al., "The sequence of the human genome," *Science*, vol. 291, no. 5507, pp. 1304–1351, 2001.
[7] F. S. Collins, M. Morgan, and A. Patrinos, "The human genome project: Lessons from large-scale biology," *Science*, vol. 300, no. 5617, pp. 286–290, 2003.
[8] M. E. Frazier, G. M. Johnson, D. G. Thomassen, C. E. Oliver, and A. Patrinos, "Realizing the potential of the genome revolution: the genomes to life program," *Science*, vol. 300, no. 5617, pp. 290–293, 2003.
[9] R. Schwartz, "Theory and algorithms for the haplotype assembly problem," *Commun. Inform. Syst.*, vol. 10, no. 1, pp. 23–38, 2010.
[10] R. Lippert, R. Schwartz, G. Lancia, and S. Istrail, "Algorithmic strategies for the single nucleotide polymorphism haplotype assembly problem," *Briefings Bioinformat.*, vol. 3, no. 1, pp. 23–31, 2002.

[11] R.-S. Wang, L.-Y. Wu, Z.-P. Li, and X.-S. Zhang, "Haplotype reconstruction from snp fragments by minimum error correction," *Bioinformatics*, vol. 21, no. 10, pp. 2456–2462, 2005.

[12] Z. Chen, B. Fu, R. Schweller, B. Yang, Z. Zhao, and B. Zhu, "Linear time probabilistic algorithms for the singular haplotype reconstruction problem from SNP fragments," *J. Comput. Biol.*, vol. 15, no. 5, pp. 535–546, 2008.

[13] S. J. Lindsay, J. K. Bonfield, and M. E. Hurles, "Shotgun haplotyping: A novel method for surveying allelic sequence variation," *Nucleic Acids Res.*, vol. 33, no. 18, pp. e152–e152, 2005.

[14] L. M. Li, J. H. Kim, and M. S. Waterman, "Haplotype reconstruction from SNP alignment," *J. Comput. Biol.*, vol. 11, nos. 2/3, pp. 505–516, 2004.

[15] J. H. Kim, M. S. Waterman, and L. M. Li, "Accuracy assessment of diploid consensus sequences," *IEEE/ACM Trans. Comput. Biol. Bioinform.*, vol. 4, no. 1, pp. 88–97, Jan.–Mar. 2007.

[16] J. H. Kim, M. S. Waterman, and L. M. Li, "Diploid genome reconstruction of ciona intestinalis and comparative analysis with ciona savignyi," *Genome Res.*, vol. 17, no. 7, pp. 1101–1110, 2007.

[17] S. Levy, G. Sutton, P. C. Ng, L. Feuk, A. L. Halpern, B. P. Walenz, N. Axelrod, J. Huang, E. F. Kirkness, G. Denisov, et al., "The diploid genome sequence of an individual human," *PLoS Biol.*, vol. 5, no. 10, p. e254, 2007.

[18] D. Aguiar and S. Istrail, "Haplotype assembly in polyploid genomes and identical by descent shared tracts," *Bioinformatics*, vol. 29, no. 13, pp. i352–i360, 2013.

[19] V. Bansal and V. Bafna, "Hapcut: An efficient and accurate algorithm for the haplotype assembly problem," *Bioinformatics*, vol. 24, no. 16, pp. i153–i159, 2008.

[20] T. M. Cover and J. Thomas, *Elements of Information Theory*. New York, NY, USA: Wiley, 1991.

[21] T. Richardson and U. Ruediger, *Modern Coding Theory*. Cambridge, U.K.: Cambridge Univ. Press, 2008.

[22] R. G. Gallager, "Low-density parity-check codes," *Res. Monograph Series*, 1963.

[23] S. Kudekar, T. J. Richardson, and R. L. Urbanke, "Threshold saturation via spatial coupling: Why convolutional LDPC ensembles perform so well over the BEC," *IEEE Trans. Inf. Theory*, vol. 57, no. 2, pp. 803–834, Feb. 2011.

[24] S. Kudekar, T. Richardson, and R. Urbanke, "Spatially coupled ensembles universally achieve capacity under belief propagation," in *Proc. IEEE Int. Symp. Inform. Theory Proc.*, 2012, pp. 453–457.

[25] S. Kumar, A. J. Young, N. Macris, and H. D. Pfister, "A proof of threshold saturation for spatially-coupled LDPC codes on BMS channels," *CoRR*, vol. abs/1301.6111, 2013.

[26] A. Montanari, "Tight bounds for LDPC and LDGM codes under MAP decoding," *IEEE Trans. Inf. Theory,*, vol. 51, no. 9, pp. 3221–3246, Sep. 2005.

[27] J. Duitama, G. K. McEwen, T. Huebsch, S. Palczewski, S. Schulz, K. Verstrepen, E.-K. Suk, and M. R. Hoehe, "Fosmid-based whole genome haplotyping of a HapMap trio child: Evaluation of single individual haplotyping techniques," *Nucleic Acids Res.*, vol. 40, no. 5, pp. 2041–2053, 2012.

[28] E. Berger, D. Yorukoglu, J. Peng, and B. Berger, "Haptree: A novel Bayesian framework for single individual polyplotyping using NGS data," *PLoS Comput Biol*, vol. 10, no. 3, p. e1003502, Mar. 2014.

[29] The 1000 Genomes Project Consortium, "A map of human genome variation from population-scale sequencing," *Nature*, vol. 467, no. 7319, pp. 1061–1073, Oct. 2010.

[30] J. Duitama, T. Huebsch, G. McEwen, E.-K. Suk, and M. R. Hoehe, "ReFHap: A reliable and fast algorithm for single individual haplotyping," in *Proc 1st ACM Int. Conf. Bioinformat. Comput. Biol.*, 2010, pp. 160–169.

[31] A. Panconesi and M. Sozio, "Fast hare: A fast heuristic for single individual SNP haplotype reconstruction," in *Proc. 4th Int. Workshop Algorithms Bioinform.*, 2004, pp. 266–277.

[32] Z. Chen, B. Fu, R. Schweller, B. Yang, Z. Zhao, and B. Zhu, "Linear time probabilistic algorithms for the singular haplotype reconstruction problem from SNP fragments," *J. Comput. Biol.*, vol. 15, no. 5, pp. 535–546, 2008.

[33] L. M. Genovese, F. Geraci, and M. Pellegrini, "SpeedHap: An accurate heuristic for the single individual SNP haplotyping problem with many gaps, high reading error rate and low coverage," *IEEE/ACM Trans. Comput. Biol. Bioinform.*, vol. 5, no. 4, pp. 492–502, Oct.–Dec. 2008.

[34] Y. Wang, E. Feng, and R. Wang, "A clustering algorithm based on two distance functions for MEC model," *Comput. Biol. Chemistry*, vol. 31, no. 2, pp. 148–150, 2007.

[35] S.-H. Kang, I.-S. Jeong, H.-G. Cho, and H.-S. Lim, "Hapassembler: A web server for haplotype assembly from snp fragments using genetic algorithm," *Biochemical Biophys. Res. Commun.*, vol. 397, no. 2, pp. 340–344, 2010.

[36] J. Duitama, G. K. McEwen, T. Huebsch, S. Palczewski, S. Schulz, K. Verstrepen, E.-K. Suk, and M. R. Hoehe, "Fosmid-based whole genome haplotyping of a HapMap trio child: Evaluation of single individual haplotyping techniques," *Nucleic Acids Res.*, vol. 40, no. 50, pp. 2041–2053, 2011.

[37] D. Aguiar and S. Istrail, "Hapcompass: a fast cycle basis algorithm for accurate haplotype assembly of sequence data," *J. Comput. Biol.*, vol. 19, no. 6, pp. 577–590, 2012.

[38] R. A. Gibbs, J. W. Belmont, P. Hardenbol, T. D. Willis, F. Yu, H. Yang, L.-Y. Ch'ang, W. Huang, B. Liu, Y. Shen, et al., "The international HapMap project," *Nature*, vol. 426, no. 6968, pp. 789–796, 2003.

Zrinka Puljiz received the BS degrees in electrical engineering in 2003 and in computing in 2004. She received the MS degree in electrical engineering in 2007 from the University of Zagreb, Croatia. She is currently working toward the PhD degree at the University of Texas at Austin. She is a member of the Wireless Communications and Networking Group (WNCG) at the University of Texas at Austin. Her research interests include wireless communications, signal processing, and bioinformatics. She is a student member of the IEEE.

Haris Vikalo received the BS degree from the University of Zagreb, Croatia, in 1995, the MS degree from Lehigh University in 1997, and the PhD degree from Stanford University in 2003, all in electrical engineering. He held a short-term appointment at Bell Laboratories, Murray Hill, NJ, in the summer of 1999. From January 2003 to July 2003, he was a postdoctoral researcher; and from July 2003 to August 2007 he was an associate scientist at the California Institute of Technology. Since September 2007, he has been with the Department of Electrical and Computer Engineering, the University of Texas at Austin, where he is currently an associate professor. He received the 2009 US National Science Foundation Career Award. His research interests include signal processing, bioinformatics, and communications. He is a senior member of the IEEE.