Optimal Haplotype Assembly via a Branch-and-Bound Algorithm

Shreepriya Das and Haris Vikalo, *Senior Member, IEEE*

***Abstract*—Haplotype assembly from high-throughput sequencing data is a computationally challenging problem. In fact, most of its formulations, including the most widely used one that relies on optimizing the minimum error correction criterion, are known to be NP-hard. Since finding exact solutions to haplotype assembly problems is difficult, suboptimal heuristics are often used. In this paper, we propose a novel method for optimal haplotype assembly that is based on depth-first branch-and-bound search of the solution space. Drawing on ideas from sphere decoding algorithms in digital communications, we exploit statistical information about errors in sequencing data to constrain the search of the haplotype space and thus efficiently find the optimal solution. Theoretical analysis and extensive simulation studies, as well as benchmarking on *1000 Genomes Project* experimental data, demonstrate efficacy of the proposed method.**

***Index Terms*—haplotype assembly, sphere decoding, depth-first branch-and-bound search, expected complexity**

I. Introduction

Tremendous advancements in high-throughput DNA sequencing technology have enabled affordable sequencing of individual genomes and opened up the possibility for routine detection and studies of genetic variations. In diploid organisms, such as humans, DNA is organized into pairs of chromosomes. Majority of the chromosome pairs are homologous, i.e., they have similar sequences that are not exactly identical but differ at a small fraction of nucleotide positions. The most common differences between two chromosomes in a homologous pair come in the form of single nucleotide polymorhisms (SNPs), isolated variants along the chromosome sequences. A pair of SNPs at the corresponding positions of the homologous chromosomes constitutes a genotype. Haplotypes are ordered collections of SNPs that are located on the same chromosome. Information about haplotypes is essential for enabling a number of personalized medicine applications. These applications include the discovery of how prone an individual is to various diseases and the search for optimal drug therapies [1], whole genome association studies [2], and studies of recombination patterns [3].

While SNP detection and genotype calling can be performed using low-throughput methods, it was high-throughput DNA sequencing that made the haploying of single individuals a reality. Each read provided by a high-throughput sequencing platform is a sample fragment of one chromosome of an individual. When a read covers multiple SNP positions, it provides information that can be used to assemble the haplotype associated with the chromosome from which the read was sampled. Today's sequencing platforms are capable of providing paired-end reads, i.e., pairs of reads that are separated by inserts of known length. This helps bridge large distances along a chromosome which is needed due to a relatively low frequency of polymorphisms – it is estimated that the SNP rate between two human chromosomes in a homologous pair is approximately 10^{-3} [4].

Haplotype assembly from error-free reads is straightforward and entails aligning reads to a reference and partitioning them in two groups, each corresponding to one of the chromo-

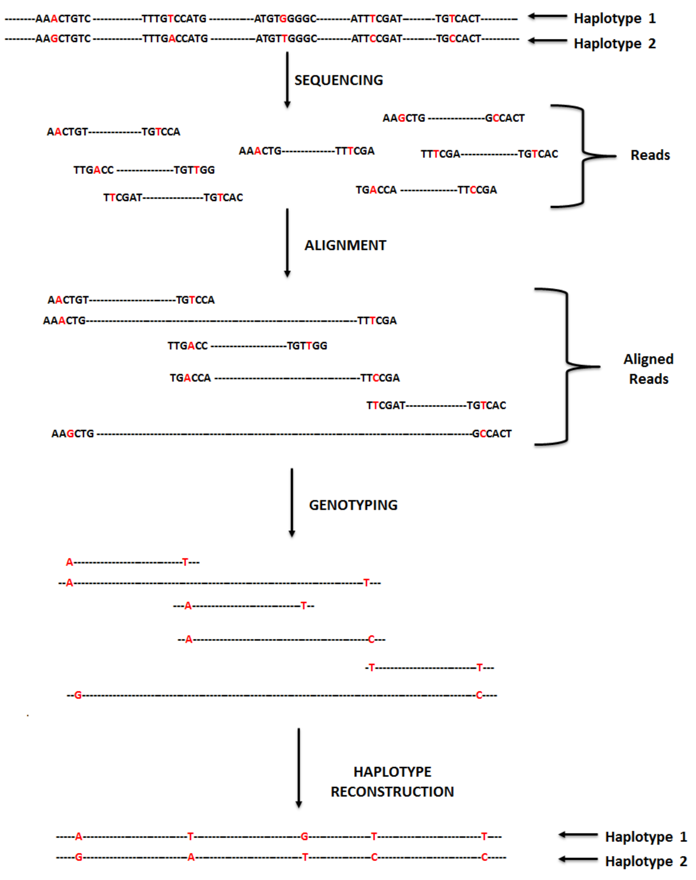

Fig. 1: *An illustration of the typical workflow in single individual haplotype reconstruction.*

S. Das is with the Division of Genetics, Department of Medicine, Brigham and Women's Hospital and Harvard Medical School, Boston, MA. H. Vikalo is with the Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX. This work was done when the first author was a PhD student in the Department of Electrical and Computer Engineering at The University of Texas at Austin. This work is funded by the NSF grant CCF 1320273.

somes in a pair. The reads that belong to a partition provide unambiguous information about the haplotype. However, sequencing is erroneous which, combined with constraints on read and insert lengths, make the haplotype assembly problem challenging. Specifically, sequencing errors cause ambiguities since the reads imperfectly sampled from a chromosome may provide conflicting information about the corresponding haplotype; moreover, errors induce uncertainty in assigning reads to chromosomes. For this reason, most of the existing haplotype assembly methods attempt to resolve error-induced ambiguities [5]. In this paper, we focus on the minimum error correction (MEC) criterion and methods that are concerned with finding the smallest number of nucleotides whose changing to a different value would resolve read assignment ambiguities. Finding the optimal solution to the MEC formulation of the haplotype assembly problem is NP-hard [5], [6].

Since the problem is NP-hard, suboptimal heuristics are often used to optimize for the MEC criterion. In [7], a greedy algorithm was proposed and applied to the first complete diploid individual genome obtained via high-throughput sequencing. In [8], an algorithm (HapCUT) that solves a max-cut reformulation of the problem was proposed and shown that it significantly outperforms the method in [7]. Bayesian frameworks relying on MCMC and Gibbs sampling schemes were proposed in [9] and [10], respectively. Recently, a greedy cut approach was proposed in [11] and applied to reads sequenced using fosmid libraries, while [12] presented a graphical approach to haplotype phasing. More recent heuristic haplotype assembly methods include a convex optimization program for minimizing the MEC score in [13] and a communication-theoretic interpretation of the problem solved via belief propagation in [14].

In addition to heuristics, computationally intensive methods for finding exact solutions to the haplotype assembly problem have been proposed. In [15], the authors used a branch-and-bound scheme to minimize the MEC objective over the space of reads, imposing a bound obtained by a random bipartition of reads. Unfortunately, exponential growth of the complexity of this scheme renders it computationally too expensive even for moderate haplotype lengths. In [16], the authors observed that, on Huref data, a large fraction of reads have small effective length which motivated their use of dynamic programming ideas for computing the optimal MEC of a revised problem that discards long reads. However, if long reads cannot be ignored, [16] abandons the search for the optimal solution and instead rephrases haplotype assembly as a MAXSAT problem and solves it using WBO and Clone solvers. Recently, [17] investigated the coverage (sample complexity) required for reconstructing haplotypes optimally with no errors.

The objective of the current paper is to demonstrate that for moderately long blocks, haplotype assembly problem can be solved exactly with a practically feasible complexity. To this end, motivated by the sphere decoding algorithm originally proposed for closest lattice point search [18] and then adopted in the field of data communications [19], [20], we propose a depth-first branch-and-bound scheme which uses statistical information about sequencing errors to impose an upper bound on the objective function and facilitate computationally efficient search for the optimal solution to the MEC formulation of the haplotype assembly problem. We analyze the expected complexity of the proposed algorithm and show that it is practically feasible for moderate lengths of haplotype blocks typically encountered in haplotyping project. Then we impose additional lower bounds on the objective function that further constrain the search space and allow even faster search for the optimal solution. The lower bounds are efficiently computed using our heuristic variable wordlength Viterbi scheme.

The paper is organized as follows. In Section II, the MEC formulation of haplotype assembly is formally stated and the new algorithm for haplotype assembly is presented, while Section III discusses its expected complexity. Section IV presents a hybrid algorithm that combines features of dynamic programming and branch-and-bound to significantly improve the speed of the basic algorithm in Section II. Experimental results on *1000 Genomes Project* dataset and simulations are discussed in Section V, while Section VI concludes the paper.

II. Haplotype assembly via branch-and-bound

The first step in haplotype assembly from high-throughput sequencing data is aligning reads to a reference, followed by the detection of polymorphic positions ("SNP calling"). Homozygous positions where all bases in all reads are identical are of no interest for the haplotype assembly problem and are thus discarded. Moreover, if a read covers only a single SNP position, it is not informative for the task at hand and the read is discarded. The remaining reads are organized into an $m \times n$ SNP fragment matrix R, where m is the number of the reads and n denotes the haplotype length. The i^{th} read is represented by the i^{th} row of R, $\mathbf{r}_i$. Diploid organisms, including humans, are typically bi-allelic which means that there may be only two possible nucleotides in any given heterozygous site of a homologous chromosome pair. These variants are labeled as -1 or 1 according to an arbitrarily chosen convention and thus a haplotype pair is conveniently represented by a pair of strings $(\mathbf{h}^1, \mathbf{h}^2)$, each of length n, with components $h_i^1, h_i^2 \in \{1, -1\}$. For the ease of notation, we introduce $\mathbf{h} = \mathbf{h}^1 = -\mathbf{h}^2$. Assume that the i^{th} read covers SNP positions $j_1, j_2, \ldots, j_k$. Then the i^{th} row of R, $\mathbf{r}_i$, will contain useful information only in k positions (in particular, positions $j_1, j_2, \ldots, j_k$). We adhere to the convention of filling all the remaining entries of $\mathbf{r}_i$ by 0's, i.e., 0's in $\mathbf{r}_i$ indicate SNP positions on the chromosome that are not covered by the i^{th}read. The start and the end of the i^{th} read are the first and the last position in $\mathbf{r}_i$ that are not 0. Note that paired-end reads typically contain substrings comprising all 0's which correspond to gaps connecting paired-end fragments; such substrings may also be present in single reads if there is missing data. Reads with a continuous string of $+1$ and -1 are gapless reads, otherwise they are referred to as gapped reads. The length of a read starting at position i and ending at position j is $j - i + 1$ (*i.e.*, it includes any possible gaps).

Remark: Note that, due to the coverage and read length limitations, available data is often insufficient for the assembly of an entire haplotype and instead enables assembly of fragmented haplotype blocks. In such situations, the data is organized in a number of SNP fragment matrices, each corresponding to one such haplotype block.

A. Problem definition

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

$$d(x,y) = \begin{cases} 1 \text{ if } x \neq 0 \text{ and } y \neq 0 \text{ and } x \neq y, \\ 0, \text{ otherwise.} \end{cases}$$

Denote the Hamming distance between read $\mathbf{r}_i$ and haplotype $\mathbf{h}$ as $\text{hd}(\mathbf{r}_i, \mathbf{h}) = \sum_{j=1}^{n} d(r_{i,j}, h_j)$, where $r_{i,j}$ and h_j denote the j^{th} components of $\mathbf{r}_i$ and $\mathbf{h}$, respectively. Then the minimum error criterion (MEC) formulation of the haplotype assembly problem is concerned with minimizing Z over $\mathbf{h}$, where the objective function

$$Z = \sum_{i=1}^{m} \min(\text{hd}(\mathbf{r}_i, \mathbf{h}), \text{hd}(\mathbf{r}_i, -\mathbf{h})), \qquad (1)$$

and m denotes the total number of reads.

B. A branch-and-bound algorithm for haplotype assembly

To minimize Z in (1) over $\mathbf{h}$, we construct and conduct a depth-first search on a tree where the node at the k^{th} level of the tree corresponds to the partial (k bases long) leading substring of $\mathbf{h}$.[1] The search tree is illustrated in Fig. 2. Recall that conflicts between reads are induced by sequencing errors and that in the MEC framework we inherently attempt to resolve conflicts by assuming the fewest possible errors. Denote the probability of an erroneous entry in R by p. Note that v, the total number of entries in R which would need to be altered so that each row in R is consistent with one of the reconstructed haplotype sequences, has a binomial distribution with cumulative mass function (cmf)

$$\mathbb{P}(v \leq k) = \sum_{j=0}^{k} \binom{\mu}{j} p^j (1-p)^{(\mu-j)},$$

where μ denotes the total number of non-zero entries in matrix R. This, in turn, can be used to impose a statistical upper bound[2] on the objective function (1). In particular, we use a depth-first search on the aforementioned binary tree to find all haploype candidates $\mathbf{h}$ satisfying

$$Z = \sum_{i=1}^{m} \min(\text{hd}(\mathbf{r}_i, \mathbf{h}), \text{hd}(\mathbf{r}_i, -\mathbf{h})) \leq b, \qquad (2)$$

where the upper bound on the number of errors, b, is computed from the binomial cmf as

$$b = \inf\{k | \sum_{j=0}^{k} \binom{\mu}{j} p^j (1-p)^{(\mu-j)} > 1 - \varepsilon\}. \qquad (3)$$

[1]A related branch-and-bound algorithm for haplotype assembly was proposed in [15] but the search there is conducted in the space of reads (of the dimension m that is typically much greater than n), does not impose statistical bounds on the objective akin to those we introduce in this section, and does not consider expected complexity of finding the solution.

[2]Similar idea has been used to enable efficient search for the closest point in a lattice in a probabilistic setting commonly encountered in data communications [19], [20].

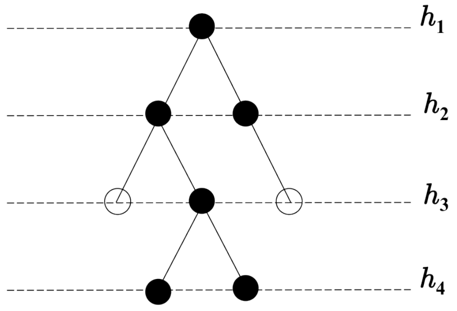

Fig. 2: *An illustration of the tree searched by the branch-and-bound algorithm. The "hollow" nodes correspond to haplotype substrings that induce costs which violate an upper bound on the objective and are hence pruned by the algorithm.*

Parameter ε determines the confidence that we will find an n-dimensional $\mathbf{h}$ which satisfies the constraint (2). For instance, setting $\varepsilon = 0.01$ means that with probability $1 - \varepsilon = 0.99$ we will find a solution to the constraint satisfaction problem (2). Clearly, while traversing close to the root of the search tree, the bound is very loose and induces little pruning. As we proceed deeper into the search tree, the bound is more frequently violated which results in discarding a large fraction of the nodes. If no $\mathbf{h}$ satisfying (2) is found, we increase the bound (i.e., reduce ε) and start the search anew.

Remark: Due to the form of the objective function (1), the search algorithm needs to simultaneously test if both $\mathbf{h}$ and $-\mathbf{h}$ are feasible haplotype sequences. For efficiency, in our implementation the algorithm traverses the tree by concatenating a partially constructed haplotype pair at the $(j-1)^{st}$ level, $\{\mathbf{h}_{1:j-1}, -\mathbf{h}_{1:j-1}\}$, with either $(-1, 1)$ or $(1, -1)$. If, for example, the algorithm traversed the first $j = 3$ levels of the search tree through $(-1, 1)$, $(-1, 1)$ and $(1, -1)$, then the partially reconstructed potential haplotype pair being tested for feasibility is $\{\mathbf{h}_{1:3}, -\mathbf{h}_{1:3}\} = \{[-1\ -1\ 1], [1,\ 1\ , -1]\}$.

C. Practical implementation issues

Note that every time a candidate $\mathbf{h}^c$ satisfying (2) is found, we can update the upper bound as $b = \sum_{i=1}^{m} \text{hd}(\mathbf{r}_i, \mathbf{h}^c)$ and proceed searching. Moreover, note that if a read covers SNPs i and k, $i < j < k$, then at the j^{th} level of the search tree we can incorporate partial contribution of the read to the cost function (1) (i.e., we do not need to wait until the k^{th} tree level to include the MEC cost induced by the read, which allows faster pruning and improves speed of the algorithm especially when dealing with long reads). The proposed algorithm is formalized as Algorithm 1.

III. Analysis of the expected complexity of the algorithm

The complexity of Algorithm 1, akin to that of motivating sphere decoding, varies over random instances of the underlying optimization problem and is hence best viewed as a random variable [19], [20]. We here characterize it by its mean for a simplified model of the haplotype assembly problem. In

Algorithm 1 Branch-and-bound algorithm for haplotype assembly of diploid chromosomes

```
Input: R, UB_{(1...n)} = 0, Z = b (upper bound)
Output: Z^*(optimal MEC), cnt (# of optimal solutions)
hap (optimal haplotypes)
Initialization: $h_1^1 = 1, h_1^2 = -1$ , $UB_{(1...n)} = Z$, $s_{(1...n)} = 0$, $obj_{(1...n)} = 0$, $k = 2$, cnt=0
Description:
1: $s_k = s_k + 1$
2: If $s_k = 1$, set $h_k^1 = 1, h_k^2 = -1$; else set $h_k^1 = -1, h_k^2 = 1$
3: $obj_k = obj_{k-1}$; softobj = 0
4: For all reads ending at level $k$, compute total MEC from these reads ($= Z_k$); set $obj_k = obj_k + Z_k$
5: For all reads starting before level k and ending after level k, compute MEC from these partial reads ($= Z_k$); softobj = softobj + $Z_k$
6: temp = $obj_k$ + softobj
7: If temp > $UB_k$, set $obj_k = 0$; Backtrack: while $s_k = 2$, set $s_k = 0$, $k = k - 1$;
8: If temp = $UB_k$ and $k = n$, cnt = cnt + 1
9: If temp < $UB_k$ and $k = n$, cnt = 1
10: If temp <= $UB_k$ and $k = n$, set $Z^* = obj_k$, set $hap^{1,2}_{(1...n)}(cnt) = h^{1,2}_{(1...n)}$, set $UB_{(1...n)} = Z^*$; Backtrack: while $s_k = 2$, set $s_k = 0$, $k = k - 1$;
11: If temp <= $UB_k$ and $k < n$, set $k = k + 1$
12: If $k > 1$, repeat steps $1 - 11$;
13: If $k = 1$ and cnt > 0 terminate algorithm, return cnt, hap, $Z^*$;
14: If $k = 1$ and cnt = 0 increase Z and restart algorithm;
```

particular, the complexity of Algorithm 1 is proportional to the number of nodes visited in the search tree in Fig. 2 and hence the expected complexity is proportional to the expected number of such nodes. The total number of nodes that survive the pruning (2) is given by

$$N = \sum_{j=1}^{n} \sum_{k=1}^{2^j} \mathcal{I}(Z_j^k \leq b), \qquad (4)$$

where b is the upper bound, $\mathcal{I}(\cdot)$ denotes an indicator function that is equal to 1 when its argument is true, and Z_j^k is the (partial) MEC score associated with the tree path ending in the k^{th} node at level j,

$$Z_j^k = \sum_{i=1}^{m} \min(\text{hd}(\mathbf{r}_{i,1:j}, \mathbf{h}_{1:j}^k), \text{hd}(\mathbf{r}_{i,1:j}, -\mathbf{h}_{1:j}^k)),$$

$\mathbf{r}_{i,1:j}$ is the vector comprising first j components of $\mathbf{r}_i$ and $\mathbf{h}_{1:j}^k$ records the collection of nodes along the partial tree search path ending in the k^{th} node at level j. The expected number of survivor nodes is then given by the expectation of (4), *i.e.*,

$$\mathbb{E}[N] = \sum_{j=1}^{n} \sum_{k=1}^{2^j} \mathbb{P}(Z_j^k \leq b), \qquad (5)$$

and the expected 'Complexity' can therefore be expressed as

$$\mathbb{E}[C] = \sum_{j=1}^{n} \sum_{k=1}^{2^j} \mathbb{P}(Z_j^k \leq b) \cdot f_p(j),$$

where $f_p(j)$ is the computational cost associated with processing a j^{th} level node and is equal to the number of non zero entries in all rows up to and including the j^{th} column. We also define the 'Complexity Exponent' as γ, where γ is given by $C = n^\gamma$ or $\gamma = \frac{log(C)}{log(n)}$.

In the general setting where inter-SNP distances are drawn from a geometric distribution and insert lengths are modeled as Gaussian random variables, finding a closed-form expression for the expected complexity of the proposed algorithm appears challenging. We therefore make simplifying assumptions and study the setting where a haplotype block of length n is sampled with m reads. A read covers no more than l SNPs but these l positions need not be contiguous (i.e. the reads can have gaps in them). We refer to l, the number of SNP locations covered by a read, as the 'effective read length'. In the following analysis, we make the further assumption that the two haplotype strands in a pair are equally likely sampled, i.e., the probabilities of a read being associated with either haplotype strand are equal. No assumption about the coverage per haplotype position is made.

Without a loss of generality, for the following argument we assume that the true haplotype pair is $\{\mathbf{h}_t, -\mathbf{h}_t\} = \{(1, \ldots 1), (-1, \cdots -1)\}$. At the j^{th} level of the search tree in Fig. 2, there are 2^j nodes possibly visited by the algorithm. For a given tree node at this level, the contributions of individual reads to Z_j^k,

$$z_{i,j}^k = \min(\text{hd}(\mathbf{r}_{i,1:j}, \mathbf{h}_{1:j}^k), \text{hd}(\mathbf{r}_{i,1:j}, -\mathbf{h}_{1:j}^k)),$$

$i = 1, 2, \ldots, m$, $j = 1, 2, \ldots, n$, $k = 1, 2, \ldots, 2^j$, are independent and hence the probability mass function (PMF) of Z_j^k, $d(Z_j^k)$, may in principle be obtained by convolving the PMFs $d(z_{i,j}^k)$. Rather than combining the PMFs by means of convolutions, however, we consider the moment generating functions (MGFs) of the relevant random variables.

The MGF of Z_j^k can be obtained as the product of the MGFs of $z_{i,j}^k$. To find the MGF of $z_{i,j}^k$, note that for a read covering no more than l SNPs (i.e., $\mathbf{r}_i$ has no more than l components from $\{-1, 1\}$), it must hold that $0 \leq z_{i,j}^k \leq \lfloor \frac{l}{2} \rfloor$. The PMF of $z_{i,j}^k$ characterizes the probabilities of $z_{i,j}^k$ taking values $0, 1, \ldots \lfloor \frac{l}{2} \rfloor$, e.g., $d_{i,j}^k(0) = d(z_{i,j}^k = 0)$, $d_{i,j}^k(1) = d(z_{i,j}^k = 1)$, etc. The corresponding MGF is given by

$$M_{z_{i,j}^k}(t) = d_{i,j}^k(0) + d_{i,j}^k(1)e^t + d_j^k(2)e^{2t} \ldots d_{i,j}^k(\lfloor \frac{l}{2} \rfloor) e^{\lfloor \frac{l}{2} \rfloor t},$$

where

$$\sum_{s=0}^{\lfloor \frac{l}{2} \rfloor} d_{i,j}^k(s) = 1.$$

A. Computing $d(z_{i,j}^k)$

Case $j = 2$: It is beneficial to first consider an illustrative scenario $j = 2$ and gapless reads that cover the first two SNPs in the haplotype. Recall our assumption that the true

l	sequence	HD	$d^k_{i,j}(0)$	$d^k_{i,j}(1)$	$d^k_{i,j}(2)$
2	(-1,-1)	0	$1-2p(1-p)$	$2p(1-p)$	
	(-1,1)	1	$2p(1-p)$	$1-2p(1-p)$	
3	(-1,-1,-1)	0	$1-3p(1-p)$	$3p(1-p)$	
	(-1,-1,1),(-1,1,-1),(-1,1,1)	1	$3p(1-p)$	$1-3p(1-p)$	
4	(-1,-1,-1,-1)	0	$(1-p)^4+p^4$	$4p(1-p)(1-2p+2p^2)$	$6p^2(1-p)^2$
	(-1,-1,-1,1),(-1,-1,1,-1),(-1,1,-1,-1),(-1,1,1,1)	1	$p(1-p)(1-2p+2p^2)$	$(1-p)^4+p^4$	$3p(1-p)(1-2p+2p^2)$
	(-1,-1,1,1),(-1,1,-1,1),(-1,1,1,-1)	2	$2p^2(1-p)^2$	$4p(1-p)(1-2p+2p^2)$	$(1-p)^4+p^4+4p^2(1-p)^2$

TABLE I: *Coefficients of the PMF $d(z^k_{i,j})$ for different effective read lengths l. For clarity, the complements of the partially reconstructed haplotype segments are omitted.*

haplotype is $\{\mathbf{h}_t, -\mathbf{h}_t\} = \{[1\ldots1], [-1\ldots-1]\}$. The pairs $\{\mathbf{h}_{1:2}, -\mathbf{h}_{1:2}\}$ corresponding to the possible paths through the nodes on the first two levels of the search tree are $\{[-1\ \ -1], [1\ 1]\}$ and $\{[1\ \ -1], [-1\ 1]\}$. The costs induced by a read either containing no errors or having both positions incorrect (which happens with the probability $(1-p)^2+p^2 = 1-2p(1-p)$) are $z^k_{i,2}=0$ and $z^k_{i,2}=1$ when traversing the paths $\{[-1\ \ -1], [1\ 1]\}$ and $\{[1\ \ -1], [-1\ 1]\}$, respectively. Likewise, the costs induced by a read containing precisely one error (which happens with the probability $2p(1-p)$) are $z^k_{i,2}=1$ and $z^k_{i,2}=0$ when traversing the paths $\{[-1\ \ -1], [1\ 1]\}$ and $\{[1\ \ -1], [-1\ 1]\}$, respectively. For convenience, let us introduce $p' = 2p(1-p)$. Therefore, the PMF of the contribution to Z^k_2 by a read that covers the first two SNPs in the haplotype is given by the mixture

$$d(z^k_{i,2}) = \frac{1}{2}\text{Bernoulli}(p') + \frac{1}{2}\text{Bernoulli}(1-p'), \quad k=1,2.$$

General case: Since $z^k_{i,j}$ depends on a path through the search tree, the coefficients in the PMF of $z^k_{i,j}$ will generally vary for two different nodes k_1 and k_2 unless $z^{k_1}_{i,j} = z^{k_2}_{i,j}$. Finding closed form expressions for $d(z^k_{i,j})$ seems challenging but numerical evaluation of the coefficients of the PMF is possible for small l, j. We illustrate the PMF coefficients for gapless reads of effective length $l=2,3,4$ in Table I.

B. Computing $d(Z^k_j)$

Case $j=2$: Continuing the illustrative example from Section III.A, it is straightforward to see that if c reads cover the first two SNP positions, then

$$d(Z^k_2) = \frac{1}{2}(B(c,p') + B(c,1-p')),$$

where $B(c,p')$ denotes the binomial distribution with parameters c and p'. Note that we here found the distribution directly, and did not need to first evaluate the moment generating function of Z^k_2 given by

$$M_{Z^k_2}(t) = \frac{1}{2}[((1-p') + p'e^t)^c + (p' + (1-p')e^t)^c].$$

General case: Given the MGFs of $z^k_{i,j}$, the MGF of Z^k_j can be computed as

$$M_{Z^k_j}(t) = \prod_{i=1}^{m}\left[d^k_{i,j}(0) + d^k_{i,j}(1)e^t + \cdots + d^k_{i,j}(\lfloor\tfrac{l}{2}\rfloor)e^{\lfloor\frac{l}{2}\rfloor t}\right].$$

In principle, we may evaluate $M_{Z^k_j}(t)$ numerically and use it to find $d(Z^k_j)$ and, consequently, evaluate (5). This, however, quickly becomes computationally challenging since for a haplotype of block length n one needs to evaluate the contributions from the MGFs of 2^n nodes. Numerical approach, therefore, may be used to provide theoretical expressions for the expected complexity of the proposed algorithm only for small lengths of haplotype blocks. In the following subsections, we consider a set of simplifying assumptions on the structure of the SNP fragment matrix that enable more efficient computation of the expected complexity.

C. Efficient evaluation for a special structure of R

Let us rearrange the reads in such a way that their starting positions are non-decreasing, i.e., if the first SNP position covered by the i^{th} read is k_i, then $k_1 \le k_2 \le \cdots \le k_m$. Let us group reads into q blocks according to their starting positions $j_1 < j_2 < \cdots < j_q$ (note that $q \le n-1$), and let w_i denotes the number of reads in the i^{th} such block. It is easy to see that if the end position of the reads in the i^{th} block (the ones starting at j_i) is the first position of the reads in the $(i+1)^{st}$ block (i.e., j_{i+1}) and the reads are gapless, then $\{Z^k_{j_i}\}$, $1 \le j_i \le q$, are mutually independent. In this scenario, we can efficiently evaluate the expected complexity for arbitrary haplotype block lengths. For clarity, we first focus on the simple scenario where all reads are of equal effective length $l=2$ and then generalize it to $l>2$.

Case $l=2$: For $l=2$, it holds that $j_i = j_{i-1}+1$, $j_0 = 0$. Let us start by considering the scenario where the number of reads in each aforementioned block is the same, i.e., $w_1 = w_2 = \cdots = w_q = w$. Then the MGF of Z^k_j, $1 \le j \le n$, is given by

$$M_{Z^k_j}(t) = \frac{1}{2^j}[(1-p'+p'e^t)^w + (p' + (1-p')e^t)^w]^j,$$

where, as before, $p' = 2p(1-p)$ is introduced for convenience. By expanding the expression within the brackets and collecting terms having the same exponent e^{kt}, the r^{th} coefficients of the PMF of Z^k_j is readily obtained as

$$\frac{1}{2^j}\binom{w}{r}[(p')^r(1-p')^{w-r} + (p')^{w-r}(1-p')^r].$$

A straightforward use of the multinomial theorem leads to the coefficients of $d(Z^k_j)$ in a closed form. Note that the support of the PMF $d(Z^k_j)$ at the j^{th} level is $\{0, 1\cdots jw\}$, and that the coefficients evaluated at the $(j-1)^{st}$ level can be used to recursively compute the coefficients at the j^{th} level. Therefore, the required coefficient can be obtained with $O(wj^2)$ time complexity (and, thus, coefficients for all n levels can be obtained with $O(wn^2)$ time complexity).

If w_i's are not all equal, the moment generating function of Z_j^k is given by

$$M_{Z_j^k}(t) = \frac{1}{2^j}\prod_{i=1}^{j}[(1-p'+p'e^i)^{w_i} + (p'+(1-p')e^t)^{w_i}],$$

where w_i is the number of reads in the i^{th} block. While the closed form expressions for the coefficients of $d(Z_j^k)$ are not available, one can find them numerically using the same recursive scheme as in the uniform coverage case. This computation can be performed with $O(\sum_{0<i,j<n} w_i w_j)$ time complexity. The support of the distribution at the n^{th} level is $\{0, 1, \cdots, \sum_{i=0}^{n} w_i\}$.

General case: We extend the previous analysis to the case where the reads are of arbitrary length but other assumptions leading to the independence of $\{Z_{j_i}^k\}$ are satisfied. In this scenario, it is sufficient to characterize the expected number of points visited at level $j_i = l-1$ and levels j that are integer multiples of $l-1$ (there is no pruning at levels that are not integer multiples of $l-1$). Unlike the $l=2$ case, here the MGF for a block containing w_1 reads is given by $M_{Z^k_{j(l-1)}}(t) =$,

$$\frac{1}{2^{l-1}}[\sum_{k=1}^{2^{l-1}}(d_{i,j}^k(0) + d_{i,j}^k(1)e^t + \cdots + d_{i,j}^k(\lfloor\frac{l}{2}\rfloor)e^{\lfloor\frac{l}{2}\rfloor t})^{w_1}],$$

$j = 1, 2 \ldots \frac{n}{l-1}$. The MGF (evaluated at levels $j(l-1)$, $j = 1, 2, \ldots, \frac{n}{l-1}$) is given by

$$\prod_{j=1}^{\frac{n}{l-1}}\frac{1}{2^{j(l-1)}}[\sum_{k=1}^{2^{j(l-1)}}(d_{i,j}^k(0)+d_{i,j}^k(1)e^t+\cdots+d_{i,j}^k(\lfloor\frac{l}{2}\rfloor)e^{\lfloor\frac{l}{2}\rfloor t})^{w_j}].$$

Note from Table I that the weights of $d(z_{i,j}^k)$ (and, therefore, the coefficients in the corresponding MGFs) depend on the tree path that the algorithm traverses. However, if two nodes k_1 and k_2 are reached by traversing paths characterized by the same minimum distance from the pair of true haplotype sequences, then the corresponding MGFs of $z_{i,j}^{k_1}$ and $z_{i,j}^{k_2}$ are equal. Therefore, for an efficient enumeration of the space of different tree paths of a specified length, one needs to examine those paths and categorize them according to their minimum Hamming distance from the pair of true haplotype sequences. The entries in Table I suggest how this enumeration may be done for partial tree search paths of length $l \in \{2, 3, 4\}$. For instance, for partial tree search paths of length $l = 3$, there are 6 out of 8 possible path sequences at the same same minimum Hamming distance from the true haplotype sequence; the remaining 2 paths coincide with the corresponding segments of one of the true haplotype sequences and thus their associated minimum Hamming distance is zero. Moreover, among the partial tree paths of length $l = 4$, there are 8 that are characterized by the same minimum Hamming distance (leading to identical MGFs of the contribution to the MEC cost by the reads of length $l = 4$), another 6 induce a different MGF, while the remaining 2 have a third MGF in common. Therefore, if the i^{th} block consists of w_i reads of length $l_i = 4$, then the i^{th} factor in the product which needs to be computed to obtain

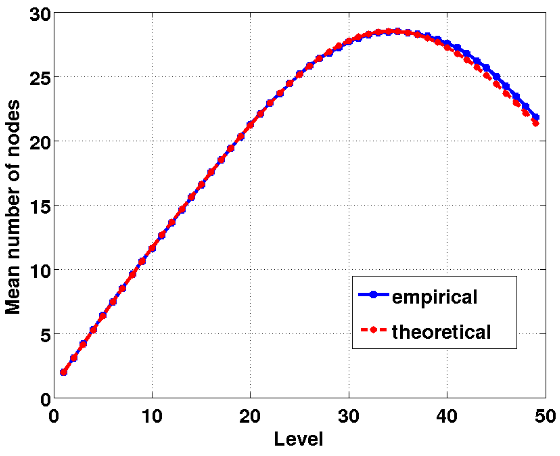

Fig. 3: *A comparison of the theoretical and empirical average number of nodes across different levels of the search tree.*

the MGF of $Z_{j_i}^k$ is given by

$$\begin{aligned}&\frac{1}{2^3}[(a_0(0) + a_0(1)e^t + a_0(2)e^{2t})^{w_i}\\ &+ 4(a_1(0) + a_1(1)e^t + a_1(2)e^{2t})^{w_i}\\ &+ 3(a_2(0) + a_2(1)e^t + a_2(2)e^{2t})^{w_i}],\end{aligned}$$

where $\{a_r(0), a_r(1), a_r(2)\}$ denotes the set of coefficients $\{d_{i,j}^k(0), d_{i,j}^k(1), d_{i,j}^k(2)\}$ in Table I for partial tree paths of length $l = 4$ being at the minimum Hamming distance HD $= r$ from the pair of true haplotype sequences, $r \in \{0, 1, 2\}$.

In Fig. 3 we compare the theoretical and empirically evaluated expected number of nodes visited at different levels of the search tree for the parameters of the problem $p = 0.01$ (the error rate in the fragment matrix R) which is the error rates in Illumina platforms), coverage $c = 10$, haplotype block length $n = 75$ and effective read length $l = 2$. In Fig. 4, we keep all the parameters the same as before but let the length of the haplotype block increase and show the expected complexity exponent as a function of n. Theoretical and experimental average complexities are displayed in Fig. 5 as a function of coverage, while Fig. 6 compares them for different error rates.

IV. Speeding up haplotype assembly using lower bounds on the MEC score

In this section, we describe a method for reducing the complexity of Algorithm 1 from Section II by performing additional pruning of the search tree in Fig. 2. In addition to the nodes pruned due to violating an upper bound in (2), additional nodes are eliminated from the search if the value of the objective they induce violates a lower bound on the MEC score. Such a lower bound needs to be computed efficiently so that the benefits of reducing the number of visited tree nodes outweigh the cost incurred by the evaluation of the bound. Here we describe the Viterbi algorithm that utilizes special structure of the SNP fragment matrix to efficiently compute lower bounds on the MEC score. To this end, we first discuss relevant properties of the objective function.

A. *Properties of the objective function Z*

Consider a data matrix R. Let R' be another read matrix obtained by performing any combination of the following operations:

(a) removing a read (i.e., a row) from the data set;
(b) truncating a read to a fixed length;
(c) deleting any non-zero entry of R; and
(d) cutting up a paired end/long read into smaller fragments.

If $\mathbf{h}^*$ optimizes the MEC objective on the read matrix R' (denoted by $Z(\mathbf{h}^*, R')$ and $\mathbf{h}^{opt}$ optimizes Z on the read matrix R, then $Z(\mathbf{h}^*, R')$ is a lower bound on $Z(\mathbf{h}^{opt}, R)$. We will utilize this property of Z throughput the section.

B. *Heuristic haplotype assembly with dynamic programming*

Here we describe an efficient dynamic programming technique that operates on a reduced data set R' to find the desired lower bound. Note that a dynamic programming solution to the haplotype assembly problem was proposed in [16]. That approach essentially employs the Viterbi algorithm to conduct a breadth-first search on a trellis with $2^{k_{max}-1}$ states, where k_{max} denotes the largest effective read length in the data set (in the case of paired-end reads, k_{max} includes the number of SNP positions covered by the insert). The trellis states correspond to all possible k_{max}-long substrings of the haplotype pairs. To ensure computational tractability of the solution, reads that are longer than a pre-selected k_{max} are discarded. Given a set of reads, the Viterbi algorithm recursively tracks the most likely sequence of states and finds the optimal path (and hence the haplotype) by backtracking through the trellis.

Let the starting position of a read be defined as the location of the first SNP within a haplotype that is covered by that read. Assume that the SNP fragment matrix R is generated using reads that are sorted according to their starting positions. The MEC scores of the states in the j^{th} trellis stage are computed using the reads having starting position j or smaller. Let $\mathcal{Z}_i$ denote the subset of reads with starting position i, $1 \leq i \leq n$, and let $z_k(i)$ denote the MEC score associated with the state k at stage i that is induced by the reads in $\mathcal{Z}_i$. Finally, let $Z_k(i)$ denote the MEC objective of the most likely state sequence which ends at $T_i = t_k$, i.e.,

$$Z_k(i) = \min_{T_1,\ldots,T_{i-1}} Z(\mathbf{r}_1, \ldots, \mathbf{r}_i, T_1, \ldots, T_{i-1}, T_i = k).$$

Then we can recursively compute $Z_k(i)$ as

$$Z_l(i+1) = \min(Z_{\lceil \frac{l}{2} \rceil}(i), Z_{2^{k_{max}-1} - \lceil \frac{l}{2} \rceil}(i)) + z_l(i+1), \quad (6)$$

where the recursion is initialized by setting $Z_0(0) = 1$, $Z_k(0) = 0$ for all $k > 0$. This recursion is at the core of the dynamic programming solution to the haplotype assembly problem, which finally backtracks through the optimal trellis path to determine the most likely sequence of states.

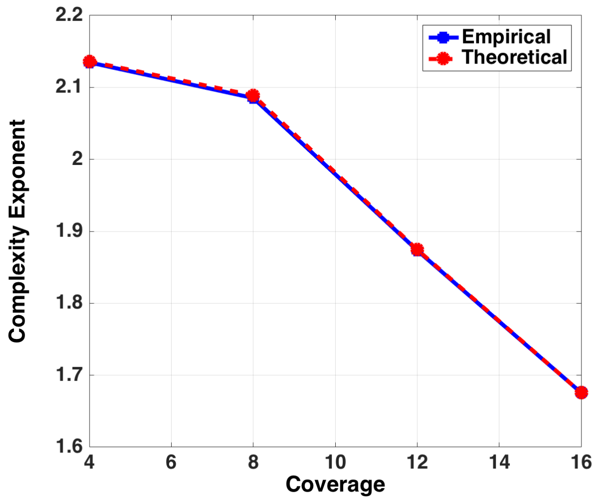

Fig. 5: *A comparison of the theoretical and empirical average complexity exponents for different sequencing coverages.*

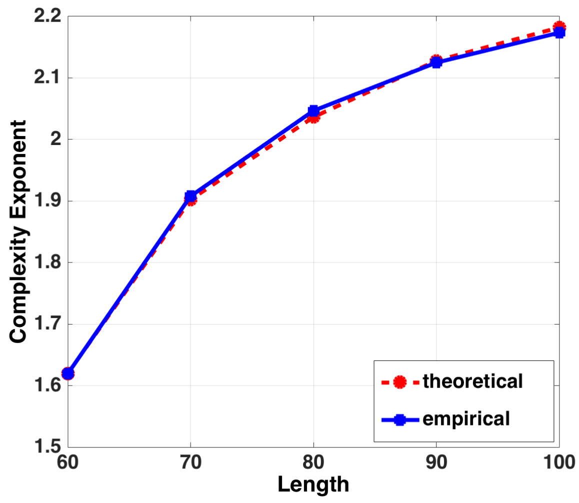

Fig. 4: *A comparison of the theoretical and empirical average complexity exponents for different lengths of haplotype blocks.*

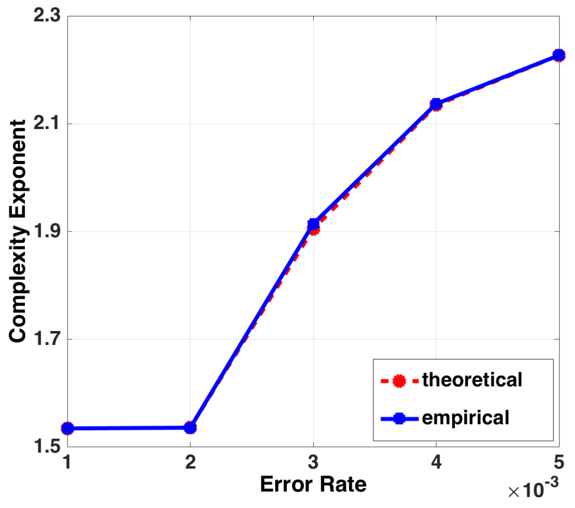

Fig. 6: *A comparison of the theoretical and empirical average complexity exponents for different error rates.*

The above described heuristic procedure does not fully exploit structure of the problem. Since real data sets contain a large fraction of reads having effective length significantly smaller than k_{max}, we propose a variable word length Viterbi algorithm alternative. In particular, assume that the longest read used in the computations of state transition costs at the i^{th} stage of the trellis has length k_i. Then the number of distinct state transition costs to be computed at stage i is exponential in k_i while the number of states is exponential in k_{max}. The variable wordlength Viterbi algorithm exploits this to reduce the complexity of the conventional Viterbi algorithm from $\mathcal{O}(\sum_{i=1}^{n} m_i k_{max} 2^{k_{max}-1})$ to $\mathcal{O}(\sum_{i=1}^{n} m_i k_i 2^{k_i-1})$.

C. Using lower bounds on the objective function to speed up branch-and-bound haplotype assembly

Since the upper bound (3) is evaluated based on the contributions of all SNP sites to the objective function Z, it may be very loose at the early stages of the search and thus induce little pruning of the nodes that are close to the root of the search tree. Relying on ideas from [21], to improve the efficiency of the search we further restrict the search space without compromising the accuracy (i.e., finding the optimal solution remains our goal). The main idea is to compute a lower bound on the value of the objective function induced by the part of the tree that is yet to be traversed. To formalize this, assume that we are visiting the k^{th} node at the j^{th} level of the search tree. The objective function (1) of our optimization can be written as $Z = Z_j^k + \bar{Z}_j^k$, where Z_j^k denotes the MEC score associated with the tree path ending in the k^{th} node at level j, while $\bar{Z}_j^k$ denotes the MEC score yet to be induced by traversing the remainder of the tree and accounting for the reads not included in the computation of Z_j^k. Thus we can restate (2) as

$$Z_j^k \leq b - \bar{Z}_j^k. \tag{7}$$

Now, if we could efficiently compute a lower bound on $\bar{Z}_j^k$, we could impose a bound at the j^{th} level of the tree that is more strict than (2) and potentially significantly reduce the number of nodes that survive the pruning while still finding the exact solution to the problem. Clearly, the benefit of computing a lower bound on $\bar{Z}_j^k$ must outweigh the additional complexity. We compute the lower bounds using the Viterbi algorithm from Section IV.B applied to a modified data matrix $R^{'}$ generated using only the reads shorter than k_{max}. A large k_{max} leads to a tight lower bound but at the cost of significant additional complexity. On the other hand, a small k_{max} keeps the complexity of finding the lower bound feasible, but the resulting bound may not enable significant node pruning.

Recall that the starting position s_j of the j^{th} read is defined as the location of the first SNP within a haplotype that is covered by that read. Additionally, we define the ending position e_j of the j^{th} read as the location of the last SNP within a haplotype covered by that read. To facilitate efficient evaluation of the lower bound at the i^{th} level of the search tree, consider a partition of the reads into the following sets:

$\mathcal{A}_i$: Set of reads $\{j\}$ such that $e_j < i$.
$\mathcal{B}_i$: Set of reads $\{j\}$ such that $e_j = i$.
$\mathcal{C}_i$: Set of reads $\{j\}$ such that $s_j < i$ and $e_j > i$.
$\mathcal{D}_i$: Set of reads $\{j\}$ such that $s_j \geq i$ and $e_j - s_j + 1 \leq k_{max}$.
$\mathcal{E}_i$: Set of reads $\{j\}$ such that $s_j \geq i$ and $e_j - s_j + 1 > k_{max}$.

Inequality (2) can then be re-written as

$$Z = Z_{\{\mathcal{A}_i,\mathcal{B}_i\}} + Z_{\{\mathcal{C}_i,\mathcal{D}_i,\mathcal{E}_i\}} \leq b, \tag{8}$$

where $Z_{\{\mathcal{A}_i,\mathcal{B}_i\}}$ and $Z_{\{\mathcal{C}_i,\mathcal{D}_i,\mathcal{E}_i\}}$ denote contributions of the reads in sets $\mathcal{A}_i \cup \mathcal{B}_i$ and $\mathcal{C}_i \cup \mathcal{D}_i \cup \mathcal{E}_i$ to Z, respectively. Following (7), we can write

$$Z_{\{\mathcal{A}_i,\mathcal{B}_i\}} \leq b - \bar{Z}_{\{\mathcal{C}_i,\mathcal{D}_i,\mathcal{E}_i\}}, \tag{9}$$

where $\bar{Z}_{\{\mathcal{C}_i,\mathcal{D}_i,\mathcal{E}_i\}}$ denotes a lower bound on $Z_{\{\mathcal{C}_i,\mathcal{D}_i,\mathcal{E}_i\}}$. Provided the lower bound is nontrivial (i.e., $\bar{Z}_{\{\mathcal{C}_i,\mathcal{D}_i,\mathcal{E}_i\}} > 0$), replacing (2) by (9) is beneficial since it would lead to eliminating more points from the search tree. Note that imposing (9) will not prune out the optimal solution since we have replaced $Z_{\{\mathcal{C}_i,\mathcal{D}_i,\mathcal{E}_i\}}$ by its *lower bound*. Clearly, the tighter (i.e., the larger) the lower bound, the more nodes will be pruned from the tree. Of course, finding the tightest possible $\bar{Z}_{\{\mathcal{C}_i,\mathcal{D}_i,\mathcal{E}_i\}}$ is infeasible since it corresponds to minimizing the MEC criterion (which is the original computationally intensive problem we started with). This motivates the use of the Viterbi algorithm on a reduced set of reads R'.

As argued in Section IV.A, certain transformations of the SNP-fragment matrix R reduce the MEC score. In particular, if a read from the sets $\mathcal{C}_i$, $\mathcal{D}_i$ or $\mathcal{E}_i$ is discarded, truncated or cut into multiple pieces, the MEC score corresponding to the new matrix R' is a lower bound on the MEC score corresponding to the original SNP-fragment matrix R. The reads belonging to the set $\mathcal{D}_i$ are shorter than k_{max} and thus need not be altered. Any paired-end/mated-pair read in $\mathcal{E}_i$ with inserts longer than $\frac{k_{max}}{2}$ is first randomly cut into 2 fragments; the resulting fragments, along with all the other reads in $\mathcal{E}_i$, are cut into smaller fragments of length $\leq k_{max}$. This procedure leads to a set of reads that are no longer than k_{max}.

Having transformed R to R', we can now employ the variable wordlength Viterbi scheme from Section IV.B to compute a lower bound that is common for all the tree points at a given level of the search tree. Clearly, the larger k_{max} we can computationally afford, the larger the fraction of non-modified reads and the tighter the resulting lower bound.

Remark: There are n different levels of the search tree at which the Viterbi algorithm could be performed. However, it may be sufficient to run the Viterbi algorithm for only a few levels in order to determine all the lower bounds. To see this, consider two levels i and j, $j < i$. Note that $card(\mathcal{D}_j) \geq card(\mathcal{D}_i)$ and $card(\mathcal{E}_j) \geq card(\mathcal{E}_i)$, where $card(\cdot)$ denotes the cardinality of its argument. Therefore, the lower bound is a decreasing function of the search tree levels. The minimum number of levels for which one needs to compute the lower bound is determined using the following procedure:

- compute the lower bounds for levels $k = 1$, $k = n$ and $k = \lceil \frac{n}{2} \rceil$.
- split the tree levels into two partitions, $1 \leq j \leq \lceil \frac{n}{2} \rceil$ and $\lceil \frac{n}{2} \rceil \leq j \leq n$, choose one of them at random and place the other one into a stack.

- if the bounds associated with the smallest and the largest levels of the selected partition are equal, the bounds for all the other levels in the partition are the same.
- if the bounds associated with the smallest and the largest levels of the partition are different, find the bound associated with the mid-level of the partition.
- repeat the above steps until all the partitions in the stack are exhausted.

The branch-and-bound algorithm for haplotype assembly with improved speed facilitated using lower bounds described in this section is formalized as Algorithm 2.

Algorithm 2 Branch-and-bound for haplotype assembly with improved speed facilitated by pruning using lower bounds

Input: R, $\text{UB}_{(1\ldots n)} = 0$, $\text{LB}_{(1\ldots n)} = 0$, $\text{Z} = b$ (upper bound)
Output: Z^*(optimal MEC), cnt (# of optimal solutions)
hap (optimal haplotypes)
Initialization: $h_1^1 = 1, h_1^2 = -1$, $\text{UB}_{(1\ldots n)} = Z - \text{LB}_{(1\ldots n)}$, $s_{(1\ldots n)} = 0$, $\text{obj}_{(1\ldots n)} = 0$, $k = 2$, cnt=0
Description:

```
1: $s_k = s_k + 1$
2: If $s_k = 1$, set $h_k^1 = 1, h_k^2 = -1$; else set $h_k^1 = -1, h_k^2 = 1$
3: $\text{obj}_k = \text{obj}_{k-1}$; softobj = 0
4: For all reads ending at level $k$, compute total MEC from these reads ($= Z_k$); set $\text{obj}_k = \text{obj}_k + \text{Z}_k$
5: For all reads starting before level k and ending after level k, compute MEC from these partial reads ($= Z_k$); $\text{softobj} = \text{softobj} + \text{Z}_k$
6: $\text{temp} = \text{obj}_k + \text{softobj}$
7: If $\text{temp} > \text{UB}_k$, set $\text{obj}_k = 0$; Backtrack: while $s_k = 2$, set $s_k = 0$, $k = k - 1$;
8: If $\text{temp} = \text{UB}_k$ and $k = n$, $\text{cnt} = \text{cnt} + 1$
9: If $\text{temp} < \text{UB}_k$ and $k = n$, $\text{cnt} = 1$
10: If $\text{temp} <= \text{UB}_k$ and $k = n$, set $\text{Z}^* = \text{obj}_k$, set $\text{hap}_{(1\ldots n)}^{1,2}(\text{cnt}) = h_{(1\ldots n)}^{1,2}$, set $\text{UB}_{(1\ldots n)} = \text{Z}^* - \text{LB}_{(1\ldots n)}$; Backtrack: while $s_k = 2$, set $s_k = 0$, $k = k - 1$;
11: If $\text{temp} <= \text{UB}_k$ and $k < n$, set $k = k + 1$
12: If $k > 1$, repeat steps $1 - 11$;
13: If $k = 1$ and $\text{cnt} > 0$ terminate algorithm, return $\text{cnt}, \text{hap}, \text{Z}^*$;
14: If $k = 1$ and $\text{cnt} = 0$ increase Z and restart algorithm;
```

V. Results and Dicussion

In this section, performance of the proposed algorithms is tested on both simulated and experimental data.

A. Performance on real datasets

We first test the performance of branch-and-bound haplotype assembly on the *1000 Genomes Project* data (individual NA12878). Table II compares the accuracy (in terms of the MEC) and runtimes of our proposed method (specifically, Algorithm 1 which turns out to be a more efficient choice) with those of HapCut and SDhaP. For SDhaP, the first shown MEC score is obtained when the algorithm assumes fully heterozygous haplotypes while the second one is obtained after detecting and eliminating homozygous SNP sites; both B&B and HapCut assume heterozygous haplotypes although a modification that deals with homozygous sites similar to that of SDhaP can readily be added. Note that while our method finds the optimal solution, for this particular data set the improvement in the MEC score over heuristics is small; however, even such small improvements in the MEC score may correspond to significant improvements in switch error rate – an important performance metric discussed in details in the next subsection.

chr#	MEC			Runtime (seconds)		
	B&B	HapCUT	SDhaP	B&B	HapCUT	SDhaP
1	2306	2317	2406(1926)	41	6	21
2	2872	2887	3002(2411)	49	8	25
3	2361	2367	2495(2005)	41	7	22
4	2605	2618	2764(2229)	44	9	25
5	2167	2179	2329(1873)	39	7	21
6	3559	3576	3716(2883)	69	12	30
7	2070	2077	2201(1721)	35	6	19
8	1838	1855	1952(1592)	35	5	19
9	1479	1489	1572(1218)	27	5	17
10	1823	1828	1945(1493)	32	5	18
11	1577	1583	1688(1320)	29	5	20
12	1589	1591	1686(1335)	28	4	15
13	1405	1414	1515(1219)	24	3	14
14	987	992	1057(832)	20	3	11
15	1061	1062	1133(886)	18	3	11
16	1263	1271	1363(1038)	22	3	13
17	1228	1236	1266(1011)	25	3	10
18	941	942	1000(796)	19	3	10
19	765	767	820(594)	12	2	8
20	794	797	831(659)	15	2	10
21	528	533	558(449)	11	1	7
22	436	438	453(354)	10	1	6

TABLE II: *MEC scores of branch-and-bound (Alg. 1 and Alg. 2), HapCUT and SDhaP for* 1000 *Genomes Project data.*

B. Performance of the algorithm on simulated data

To thoroughly test the proposed algorithms under different performance criteria, we generate multiple synthetic datasets emulating various experimental settings. To simulate a sequencing process, we randomly generate reads with starting positions randomly selected across the genome. We simulate different coverages, different sequencing error rates and different block lengths.

Fig. 7 shows the expected complexity exponent as a function of block length for our proposed algorithms at 3 different coverages - 10, 20 and 30 – for the data error rate of 0.01. As can be seen from the figure, lower bounding prevents exponential complexity growth otherwise evident in the branch-and-bound scheme that uses no lower bounding speed-up. For shorter blocks, the higher complexity of the scheme that relies on lower bounding technique is due to the additional $O(2^{k_{max}}n^2)$ operations needed to compute the lower bounds. Fig. 8 shows the expected complexity for 3 different error rates at a fixed coverage $c = 10$. For the error rate of 0.01, the expected complexity of Algorithm 1 deteriorates rapidly while that of Algorithm 2 changes little over the considered range of parameters. For the low data error rates of 0.001 and 0.003, benefits of additional pruning do not overcome the additional

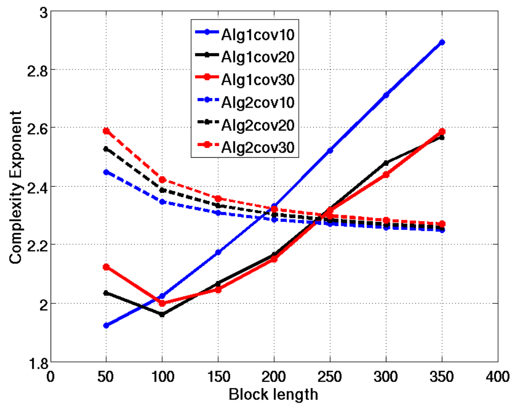

Fig. 7: *The expected complexity exponent for the branch-and-bound with (Alg. 2) and without (Alg. 1) lower bounding as a function of block lengths for coverages* 10, 20 *and* 30.

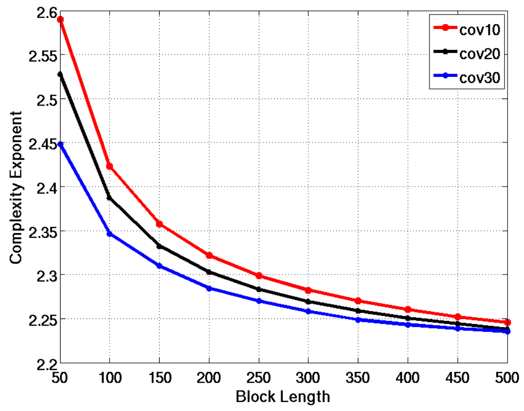

Fig. 9: *The expected complexity exponent for Algorithm 2 as a function of block lengths for coverages* 10, 20 *and* 30. *Block lengths of up to* 500 *can be assembled with sub-cubic complexity.*

cost of computing the lower bounds and thus Algorithm 1 is the preferred choice in this scenario.

Fig. 9 further explores the expected complexity exponent of Algorithm 2, showing it as a function of the block length for the data error rate 0.003 and 3 different coverages (10, 20 and 30). As seen from the figure, even for the block lengths of 500 the expected complexity is sub-cubic. Fig. 10 shows the impact of data error rate on the expected complexity exponent. As can be seen, for error rates 0.003 and 0.01, and blocks of length smaller than 500, the expected complexity exponent is low. However, for the higher error rate of 0.03, the expected complexity exponent starts growing exponentially as the block length exceeds 250. Therefore, haplotype assembly of moderately long block at error rates typically encountered in state-of-the-art sequencing platforms can be solved with sub-cubic complexity.

A crucial accuracy metric of haplotype assembly is the switch error rate (SWER), defined as the number of switches (recombination events in the inferred phased haplotypes) that are required to obtain the true haplotype phase. This comparison is typically expressed as a rate: the number of switches required divided by the number of opportunities for switch error, which is the number of heterozygote markers in the individual's genotype minus 1 (the first heterozygote marker can be assigned an arbitrary phase). Such a comparison may be obtained here since the ground truth is known for the simulated datasets. Fig. 11 shows the SWER of Algorithm 2 for different error rates as a function of block length for a fixed coverage ($c = 10$). As expected, the SWER increases with the data error rates. There is a small increase in the SWER as the block get longer. For a comparison, we also show the SWER obtained by phasing using HapCUT. The suboptimal solutions obtained

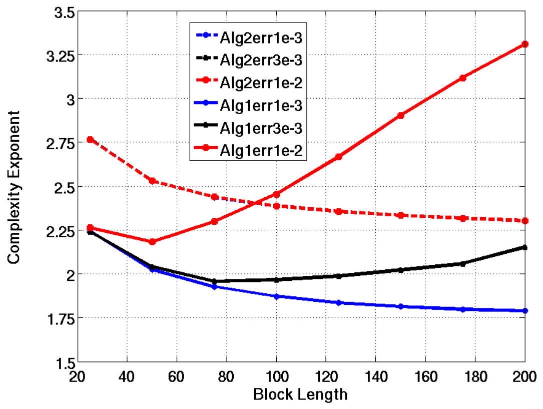

Fig. 8: *The expected complexity exponents for Algorithm 1 and Algorithm 2 as a function of block lengths for error rates of* 0.001, 0.003 *and* 0.01. *The complexity of Algorithm 2 is dominated by the Viterbi algorithm used for finding lower bounds and thus the three dashed plots seemingly coincide (since the complexity of the Viterbi algorithm does not depend on the data error rates).*

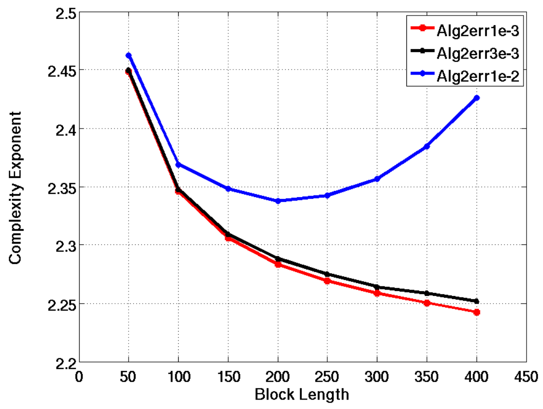

Fig. 10: *The expected complexity exponent for Algorithm 2 as a function of block lengths for error rates of* 0.003, 0.01 *and* 0.03. *For error rates of* 0.03 *and block lengths larger than* 200, *the complexity starts growing exponentially.*

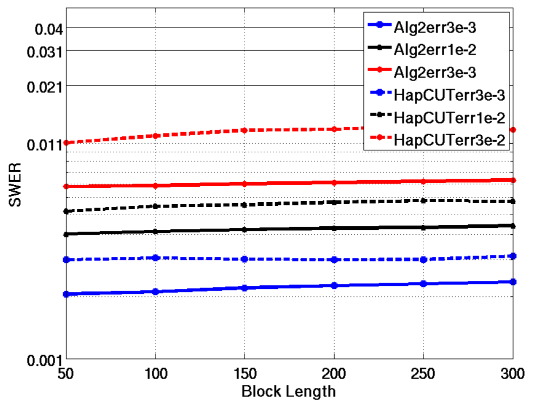

Fig. 11: *The switch error rates of Algorithm 2 and HapCUT as a function of block lengths for error rates of* 0.003*,* 0.01 *and* 0.03*.*

via HapCUT lead to worse SWER than that of branch-and-bound solutions. Fig. 12 shows the SWER as a function of coverage (10, 15 and 20) for a fixed error rate (0.01). Here the impact of coverage is more noticeable – there is almost an order of improvement as the coverage is increased from 10 to 15 and from 15 to 20. The SWERs of HapCUT are also shown, demonstrating improvement of our proposed technique over the existing method.

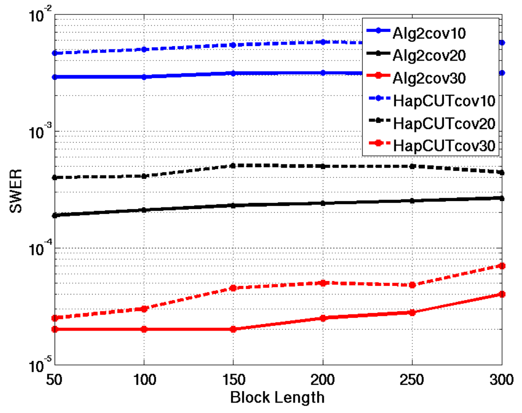

Fig. 12: *The switch error rates of Algorithm 2 and HapCUT as a function of block lengths for coverages* 10*,* 15 *and* 20*.*

VI. Conclusion

We proposed a novel depth-first branch-and-bound algorithm for finding optimal solution to the single individual haplotyping problem, and developed its fast variant. The proposed algorithms exploit structure inherent to the haplotype assembly problem and statistical information about sequencing errors to facilitate computationally efficient search for the solution to the minimum error correction formulation of haplotype assembly. We characterized expected complexity of the proposed method and demonstrated its practical feasibility on both simulated and real data. In scenarios typically encountered when processing high-throughput sequencing data, the proposed method finds the optimal solution to the haplotype assembly problem with low expected complexity. As part of the future work, it is of interest to analytically characterize the probability of error of the algorithm (e.g., by taking an approach similar to the one in [22]), and to extend the presented ideas to the assembly of polyploid haplotypes.

References

[1] A. G. Clark, "The role of haplotypes in candidate gene studies," *Genetic epidemiology*, vol. 27, no. 4, pp. 321–333, 2004.

[2] 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.

[3] P. C. Sabeti, D. E. Reich, J. M. Higgins, H. Z. Levine, D. J. Richter, S. F. Schaffner, S. B. Gabriel, J. V. Platko, N. J. Patterson, G. J. McDonald *et al.*, "Detecting recent positive selection in the human genome from haplotype structure," *Nature*, vol. 419, no. 6909, pp. 832–837, 2002.

[4] R. Sachidanandam, D. Weissman, S. C. Schmidt, J. M. Kakol, L. D. Stein, G. Marth, S. Sherry, J. C. Mullikin, B. J. Mortimore, D. L. Willey *et al.*, "A map of human genome sequence variation containing 1.42 million single nucleotide polymorphisms," *Nature*, vol. 409, no. 6822, pp. 928–933, 2001.

[5] G. Lancia, V. Bafna, S. Istrail, R. Lippert, and R. Schwartz, "Snps problems, complexity, and algorithms," in *Algorithms—ESA 2001*. Springer, 2001, pp. 182–193.

[6] R. Cilibrasi, L. Van Iersel, S. Kelk, and J. Tromp, "On the complexity of several haplotyping problems," in *Algorithms in Bioinformatics*. Springer, 2005, pp. 128–139.

[7] 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 biology*, vol. 5, no. 10, p. e254, 2007.

[8] 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.

[9] 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.

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

[11] 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 research*, p. gkr1042, 2011.

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

[13] S. Das and H. Vikalo, "SDhaP: Haplotype assembly for diploids and polyploids via semi-definite programming," *BMC Genomics*, vol. 16:260, April 2015.

[14] Z. Puljiz and H. Vikalo, "Decoding genetic variations: Communications-inspired haplotype assembly," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2016.

[15] 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.

[16] D. He, A. Choi, K. Pipatsrisawat, A. Darwiche, and E. Eskin, "Optimal algorithms for haplotype assembly from whole-genome sequence data," *Bioinformatics*, vol. 26, no. 12, pp. i183–i190, 2010.

[17] Y. Chen, G. Kamath, C. Suh, and D. Tse, "Community recovery in graphs with locality," June 2016, arXiv:1602.03828.

[18] U. Fincke and M. Pohst, "Improved methods for calculating vectors of short length in a lattice, including a complexity analysis," *Math. Comp.*, vol. 44, pp. 463–471, 1985.

[19] H. Vikalo, "Sphere decoding algorithms for digital communications," Ph.D. dissertation, Stanford University, 2002.

[20] B. Hassibi and H. Vikalo, "Maximum-likelihood decoding and integer least-squares: The expected complexity," in *Multiantenna Channels: Capacity, Coding and Signal Processing*, J. Foschini and S. Verdu, Eds. American Mathematical Society (AMS), 2003, pp. 161 – 191.

[21] M. Stojnic, H. Vikalo, and B. Hassibi, "Speeding up the sphere decoder with H-infinity and SDP inspired lower bounds," *IEEE Transactions on Signal Processing*, vol. 56, no. 2, pp. 712 – 726, February 2008.

[22] M. El-Khamy, H. Vikalo, B. Hassibi, and R. J. McEliece, "Performance of sphere decoding of block codes," *IEEE Transactions on Communications*, vol. 57, no. 10, pp. 2940–2950, 2009.