Bioinformatics, 32(11), 2016, 1610–1617
doi: 10.1093/bioinformatics/btv495
Advance Access Publication Date: 26 August 2015
Original Paper

OXFORD

Genome analysis

HapCol: accurate and memory-efficient haplotype assembly from long reads

Yuri Pirola[1,†], Simone Zaccaria[1,†], Riccardo Dondi[2], Gunnar W. Klau[3,4], Nadia Pisanti[4,5] and Paola Bonizzoni[1,*]

[1]Dipartimento di Informatica Sistemistica e Comunicazione (DISCo), Univ. degli Studi di Milano-Bicocca, Milan, Italy, [2]Dipartimento di Scienze Umane e Sociali, Univ. degli Studi di Bergamo, Bergamo, Italy, [3]Life Sciences group, Centrum Wiskunde & Informatica (CWI), Amsterdam, The Netherlands, [4]ERABLE Team, INRIA, Lyon, France and [5]Dipartimento di Informatica, Univ. degli Studi di Pisa, Pisa, Italy

*To whom correspondence should be addressed.
[†]The authors wish it to be known that, in their opinion, the first two authors should be regarded as joint First Authors.
Associate Editor: Gunnar Ratsch

Received on April 7, 2015; revised on July 24, 2015; accepted on August 10, 2015

Abstract

Motivation: *Haplotype assembly* is the computational problem of reconstructing haplotypes in diploid organisms and is of fundamental importance for characterizing the effects of single-nucleotide polymorphisms on the expression of phenotypic traits. Haplotype assembly highly benefits from the advent of 'future-generation' sequencing technologies and their capability to produce long reads at increasing coverage. Existing methods are not able to deal with such data in a fully satisfactory way, either because accuracy or performances degrade as read length and sequencing coverage increase or because they are based on restrictive assumptions.

Results: By exploiting a feature of future-generation technologies—the uniform distribution of sequencing errors—we designed an exact algorithm, called HapCol, that is exponential in the maximum number of corrections for each single-nucleotide polymorphism position and that minimizes the overall error-correction score. We performed an experimental analysis, comparing HapCol with the current state-of-the-art combinatorial methods both on real and simulated data. On a standard benchmark of real data, we show that HapCol is competitive with state-of-the-art methods, improving the accuracy and the number of phased positions. Furthermore, experiments on realistically simulated datasets revealed that HapCol requires significantly less computing resources, especially memory. Thanks to its computational efficiency, HapCol can overcome the limits of previous approaches, allowing to phase datasets with higher coverage and without the traditional all-heterozygous assumption.

Availability and implementation: Our source code is available under the terms of the GNU General Public License at http://hapcol.algolab.eu/.

Contact: bonizzoni@disco.unimib.it

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Diploid organisms such as humans contain two sets of chromosomes, one from each parent. Reconstructing the two distinct copies of each chromosome, called *haplotypes*, is crucial for characterizing the genome of an individual. The process is known as *phasing* or *haplotyping* and the provided information may be of fundamental importance for many applications, such as analyzing the relationships between genetic variation and gene function, or between genetic variation and disease susceptibility (Browning and Browning, 2011; Duitama *et al.*, 2012). In diploid species, haplotyping requires

assigning the variants to the two parental copies of each chromosome, which exhibit differences in terms of *single-nucleotide polymorphisms* (SNPs). Since a large scale direct experimental reconstruction of the haplotypes from the collected samples is not yet cost-effective (Kuleshov *et al.*, 2014), a computational approach—called *haplotype assembly*—that considers a set of reads, each one sequenced from a chromosome copy, has been proposed. Reads (also called *fragments*) have to be assigned to the unknown haplotypes, using a reference genome in a preliminary mapping phase, if available. This involves dealing in some way with sequencing and mapping errors and leads to a computational task that is generally modelled as an optimization problem (Lancia *et al.*, 2001; Lippert *et al.*, 2002).

Minimum error correction (MEC) (Lippert *et al.*, 2002) is one of the prominent combinatorial approaches for haplotype assembly. It aims at correcting the input data with the minimum number of corrections to the SNP values, such that the resulting reads can be unambiguously partitioned into two sets, each one identifying a haplotype. wMEC (Greenberg *et al.*, 2004) is the weighted variant of the problem, where each possible correction is associated with a weight that represents the confidence degree assigned to that SNP value at the corresponding position. This confidence degree is a combination of the probability that an error occurred during sequencing (phred-based error probability) for that base call and of the confidence of the read mapping to that genome position. The usage of such weights has been experimentally validated as a powerful way to improve accuracy (Zhao *et al.*, 2005).

Haplotype assembly benefits from technological developments in genome sequencing. In fact, the advent of next-generation sequencing (NGS) technologies provided a cost-effective way of assembling the genome of diploid organisms. However, to assemble accurate haplotypes, it is necessary to have reads that are long enough to span several different heterozygous positions (Duitama *et al.*, 2012). This kind of data is becoming increasingly available with the advent of 'future-generation' sequencing technologies such as single molecule real-time technologies like PacBio RS II (http://www.pacificbiosciences.com/products/) and Oxford Nanopore flow cell technologies like MinION (https://www.nanoporetech.com/). These technologies, thanks to their ability of producing single end reads longer than 10 000 bases, eliminate the need of paired-end data and have already been used for tasks like genome finishing and haplotype assembly (Smith *et al.*, 2012). Besides read length, the future-generation sequencing technologies produce fragments with novel features, such as the uniform distribution of sequencing errors, that are not properly addressed (or exploited) in most of the existing methods that, instead, are tailored to the characteristics of traditional NGS technologies.

Recently, MEC and wMEC approaches have been used in the context of long reads, confirming that long fragments allow to assemble haplotypes more accurately than traditional short reads (Aguiar and Istrail, 2012; Duitama *et al.*, 2012; Patterson *et al.*, 2014, 2015). Since MEC is NP-hard (Cilibrasi *et al.*, 2007), exact solutions have exponential complexity. Different approaches tackling the computational hardness of the problem have been proposed in literature. Integer linear programming techniques have been recently used (Chen *et al.*, 2013), but the approach failed to optimally solve some 'difficult blocks'. There were also proposed fixed-parameter tractable (FPT) algorithms that take time exponential in the number of variants per read (Bonizzoni *et al.*, 2015; He *et al.*, 2010, 2013) and, hence, are well-suited for short reads but become unfeasible for long reads. For this kind of data, heuristic approaches have been proposed to respond to the lack of exact solutions (Bansal and Bafna, 2008; Duitama *et al.*, 2012). Most of the proposed heuristics, such as ReFHap (Duitama *et al.*, 2010), make use of the traditional *all-heterozygous* assumption, that forces the heterozygosity of all the phased positions. These heuristics have good performances but do not offer guarantees on the optimality of the returned solution (Duitama *et al.*, 2012). Two recent articles (Kuleshov, 2014; Patterson *et al.*, 2014) aim at processing future-generation long reads by introducing algorithms exponential in the sequencing coverage, a parameter which is not expected to grow as fast as read length with the advent of future-generation technologies. The first algorithm, called ProbHap (Kuleshov, 2014), is a probabilistic dynamic programming algorithm that optimizes a likelihood function generalizing the objective function of MEC. Albeit ProbHap is significantly slower than the previous heuristics, it obtained a noticeable improvement in accuracy. The second approach, called WhatsHap (Patterson *et al.*, 2014), is the first exact algorithm for wMEC that is able to process long reads. It was shown to be able to obtain a good accuracy on simulated data of long reads at coverages up to 20× and to outperforms all the previous exact approaches. However, it cannot handle coverages higher than 20×, and its performance evidently decreases when approaching that limit.

In this article, we exploit a characteristic of future-generation technologies, namely the uniform distribution of sequencing errors, for introducing (Section 2) an exact FPT algorithm for a new variant, called *k*-cMEC, of the wMEC problem where the parameters are (i) the maximum number k of corrections that are allowed on each SNP position and (ii) the coverage. The new algorithm, called HapCol, is based on a characterization of feasible solutions given in Bonizzoni *et al.* (2015) and its time complexity is $O(\text{cov}^{k+1}Lm)$ (albeit it is possible to prove a stricter bound), where cov is the maximum coverage, L is the read length and m is the number of SNP positions. HapCol is able to work without the all-heterozygous assumption.

In Section 3, we experimentally compare accuracy and performance of HapCol on real and realistically simulated datasets with three state-of-the-art approaches for haplotype assembly—ReFHap, ProbHap and WhatsHap. On a real standard benchmark of long reads (Duitama *et al.*, 2012), we executed each tool under the all-heterozygous assumption, since this dataset has low coverage (∼3× on average) and since the covered positions are heterozygous with high confidence. HapCol turns out to be competitive with the considered methods, improving the accuracy and the number of phased positions. We also assessed accuracy and performance of HapCol on a large collection of realistically simulated datasets reflecting the characteristics of 'future-generation' sequencing technologies that are currently (or soon) available (coverage up to 25×, read length from 10 000 to 50 000 bases, substitution error rate up to 5% and indel rate equal to 10%) (Carneiro *et al.*, 2012; Jain *et al.*, 2015; Roberts *et al.*, 2013). When considering higher coverages, interesting applications such as SNP calling or heterozygous SNPs validation become feasible and reliable (Nielsen *et al.*, 2011). Since these applications require that haplotypes are reconstructed without the all-heterozygous assumption, on the simulated datasets we only considered the tools that do not rely on this assumption—WhatsHap and HapCol. Results on the simulated datasets with coverage 15–20× show that HapCol, while being as accurate as WhatsHap (they achieve an average error of ∼2%), is faster and significantly more memory efficient (∼2 times faster and ∼28 times less memory). The efficiency of HapCol allows to further improve accuracy. Indeed, the experimental results show that HapCol is able to process datasets with coverage 25× on standard workstations/small servers (whereas WhatsHap exhausted all the available memory, 256 GB) and that, since the number of ambiguous/uncalled positions

decreases, the haplotypes reconstructed by HapCol at coverage 25× are ∼9% more accurate than those reconstructed at coverage 20×.

2 Methods

2.1 Preliminary definitions

Let s be a vector. Then, we denote the value of s at position t by $s[t]$. A *haplotype* is a vector h of length m belonging to $\{0,1\}^m$. Let h_1, h_2 be the two haplotypes of an individual. A position j is called *heterozygous* if $h_1[j] \neq h_2[j]$, otherwise (i.e. if $h_1[j] = h_2[j]$) j is called *homozygous*. A *fragment* is a vector f of length m belonging to $\{0,1,-\}^m$. In a fragment f, a position $f[j] = -$ is called a *hole*. A *gap* in a fragment f is a maximal sub-vector of f of holes, preceded and followed by a non-hole element. Moreover, the length of a fragment f is defined as the number of elements contained in f between the leftmost and rightmost non-hole elements (included).

A *fragment matrix* is a matrix $\mathcal{M}$ consisting of n rows (fragments) and m columns (SNPs). We indicate as L the maximum length for all the fragments i in $\mathcal{M}$. We denote by $\mathcal{M}_j$ the jth column of $\mathcal{M}$. Notice that each column of $\mathcal{M}$ is a vector in $\{0,1,-\}^n$, while each row is a vector in $\{0,1,-\}^m$.

Given two row vectors s_1 and s_2 belonging to $\{0,1,-\}^m$, s_1 and s_2 are in *conflict* when there exists a position j, with $1 \leq j \leq m$, such that $s_1[j] \neq s_2[j]$ and $s_1[j], s_2[j] \neq -$, otherwise s_1 and s_2 are in *agreement*. A fragment matrix $\mathcal{M}$ is *conflict free* if and only if there exist two haplotypes h_1, h_2 such that each row of $\mathcal{M}$ is in agreement with one of h_1 and h_2. In an equivalent way, a fragment matrix $\mathcal{M}$ is conflict free if and only if there exists a *bipartition* (P_1, P_2) of the fragments in $\mathcal{M}$ such that each pair of fragments in P_1 is in agreement and each pair of fragments in P_2 is in agreement. A *correction* of the entry $\mathcal{M}_j[i]$, where $\mathcal{M}_j[i] \neq -$, is a flip of the value of $\mathcal{M}_j[i]$. Now we are able to introduce the MEC problem.

Problem 1. MEC (Lippert *et al.*, 2002)
Input: a matrix $\mathcal{M}$ of fragments.
Output: a conflict free matrix $\mathcal{M}'$ obtained from $\mathcal{M}$ with the minimum number of corrections.

A column of a matrix is called *homozygous* if it contains values in $\{0,-\}$ or in $\{1,-\}$, otherwise it is called *heterozygous*. We say that a fragment i is *active* on a column $\mathcal{M}_j$, if $\mathcal{M}_j[i] = 0$ or $\mathcal{M}_j[i] = 1$. The *active fragments* of a column $\mathcal{M}_j$ are the set $\text{active}(\mathcal{M}_j) = \{i : \mathcal{M}_j[i] \neq -\}$. The *coverage* of the column $\mathcal{M}_j$ is defined as the number cov_j of fragments that are active on $\mathcal{M}_j$, that is $\text{cov}_j = |\text{active}(\mathcal{M}_j)|$. In the following, we indicate as cov the maximum coverage over all the columns in $\mathcal{M}$. Given two columns $\mathcal{M}_{j_1}$ and $\mathcal{M}_{j_2}$, we denote by $\text{active}(\mathcal{M}_{j_1}, \mathcal{M}_{j_2})$ the intersection $\text{active}(\mathcal{M}_{j_1}) \cap \text{active}(\mathcal{M}_{j_2})$. Notice that on the one hand, any heterozygous column $\mathcal{M}_j$ encodes a bipartition of the fragments in $\text{active}(\mathcal{M}_j)$ indicating which one belongs to h_1 and which one belongs to h_2. On the other hand, any homozygous column $\mathcal{M}_j$ does not encode a specific bipartition and, since it gives no information on how its active fragments have to be partitioned, it is 'in accordance' with any other bipartition or heterozygous column.

Definition 1: Two columns $\mathcal{M}_{j_1}, \mathcal{M}_{j_2}$ of a fragment matrix $\mathcal{M}$ are in *accordance* if (1) at least one of $\mathcal{M}_{j_1}, \mathcal{M}_{j_2}$ is homozygous or (2) $\mathcal{M}_{j_1}, \mathcal{M}_{j_2}$ are both heterozygous and on $\text{active}(\mathcal{M}_{j_1}, \mathcal{M}_{j_2})$ they are identical or complementary.

The *correction distance* between two columns $\mathcal{M}_{j_1}, \mathcal{M}_{j_2}$ evaluates the minimum number of corrections needed to transform $\mathcal{M}_{j_1}$ and $\mathcal{M}_{j_2}$ into heterozygous columns in accordance and it is defined as $d(\mathcal{M}_{j_1}, \mathcal{M}_{j_2}) = \min\{|E|, |\overline{E}|\}$, where $E = \{i : \mathcal{M}_{j_1}[i] \neq \mathcal{M}_{j_2}[i] \wedge \mathcal{M}_{j_1}[i] \neq - \wedge \mathcal{M}_{j_2}[i] \neq -\}$ and $\overline{E} = \{i : \mathcal{M}_{j_1}[i] = \mathcal{M}_{j_2}[i] \wedge \mathcal{M}_{j_1}[i] \neq - \wedge \mathcal{M}_{j_2}[i] \neq -\}$. Given a column $\mathcal{M}_j$ of a fragment matrix $\mathcal{M}$, we define the *homozygous distance* $H(\mathcal{M}_j)$ as the number of times the minor allele (i.e. the least frequent value of the column) appears in $\mathcal{M}_j$ if it is not greater than an integer k, or infinity otherwise. More formally, $H(\mathcal{M}_j)$ is equal to $d(\mathcal{M}_j, \underline{0})$ if $d(\mathcal{M}_j, \underline{0}) \leq k$ (notice that $d(\mathcal{M}_j, \underline{1}) = d(\mathcal{M}_j, \underline{0})$, where $\underline{0}$ and $\underline{1}$ are the columns composed only of zeros and ones, respectively) or to ∞ otherwise. Homozygous columns cannot induce a conflict due to the fact that the corresponding positions in the two reconstructed haplotypes can be homozygous with no influence on the other positions. For this reason, we can remove every homozygous column from any input fragment matrix $\mathcal{M}$ without changing the optimal solution and we can assume that $\mathcal{M}$ is only composed of heterozygous columns. However, notice that a heterozygous column $\mathcal{M}_j$ in the input can be transformed into a homozygous column $\mathcal{M}'_j$ in the output. As a consequence, the optimal solution $\mathcal{M}'$ can potentially contain homozygous columns. Furthermore, given a conflict free matrix $\mathcal{M}'$, notice that the two resulting haplotypes h_1, h_2 can be easily computed from the bipartition of the fragments induced by the columns of $\mathcal{M}'$.

In the weighted variant wMEC of MEC, there is a weight $w(\mathcal{M}_j[i])$ associated with each non-hole entry $\mathcal{M}_j[i]$ of the input matrix $\mathcal{M}$ that represents the cost of correcting that entry. In this case, the goal is to minimize the total weight instead of the number of corrections.

Each gap in any fragment of the input matrix $\mathcal{M}$ can be modeled as zero-weight entries equal to 0 or 1. For this reason, even though we propose an approach that considers fragment matrices without gaps, called *gapless fragment matrices*, the approach can be easily extended to deal with any general fragment matrix $\mathcal{M}$.

Lemma 1 (Bonizzoni *et al.*, 2015) proves a property of these matrices that will be fundamental for our FPT algorithm.

Lemma 1: Consider a gapless fragment matrix $\mathcal{M}$. Then, $\mathcal{M}$ is conflict free if and only if each pair of columns is in accordance.

2.2 The k-constrained MEC problem

In this work, we introduce a variant of the MEC problem, called k-cMEC, motivated by the uniform distribution of sequencing errors of future-generation technologies, where the number of errors (hence, corrections) per column are bounded by an integer k. Given an input fragment matrix $\mathcal{M}$, a conflict free fragment matrix $\mathcal{M}'$ obtained from $\mathcal{M}$ with h corrections is defined as a *k-corrected matrix* for $\mathcal{M}$ if for each column $\mathcal{M}_j$ we have $d(\mathcal{M}_j, \mathcal{M}'_j) \leq k$. According to this definition we introduce the following variant of MEC:

Problem 2. k-constrained MEC (k-cMEC)
Input: a fragment matrix $\mathcal{M}$ and an integer k.
Output: a k-corrected matrix $\mathcal{M}'$ for $\mathcal{M}$ obtained with the minimum number of corrections.

Given a k-corrected matrix $\mathcal{M}'$ for a fragment matrix $\mathcal{M}$, we can see each heterozygous column $\mathcal{M}'_j$ in $\mathcal{M}'$ as the correction of the corresponding column $\mathcal{M}_j$ in $\mathcal{M}$. Hence, considering a column $\mathcal{M}_j$, we define a k-correction B_j for $\mathcal{M}_j$ as a vector in $\{0,1,-\}^n$ with $\text{active}(B_j) = \text{active}(\mathcal{M}_j)$ such that $d(\mathcal{M}_j, B_j) \leq k$ and B_j is heterozygous. According to this definition, a k-correction B_j describes a feasible way to transform $\mathcal{M}_j$ into the heterozygous column $\mathcal{M}'_j$ when $d(\mathcal{M}'_j, B_j) = 0$. Therefore, we define the space of these corrections as β_j, such that β_j is the set containing all the possible k-corrections B_j

for the column $\mathcal{M}_j$. Notice that $\underline{0}$ and $\underline{1}$ can be imagined as the corrections for any homozygous column in $\mathcal{M}'$.

The weighted variant of this problem can be easily defined in the same way as wMEC for MEC. The goal of the weighted version is to compute a k-corrected matrix $\mathcal{M}'$ obtained from $\mathcal{M}$ with minimum total weight.

Consider a fragment matrix $\mathcal{M}$. There always exists a feasible solution for the MEC problem on input $\mathcal{M}$, while a feasible solution for the k-cMEC problem, for a fixed k, on input $\mathcal{M}$ may not exist. This implies that a feasible solution for the MEC problem on input $\mathcal{M}$ may not be a feasible solution for the k-cMEC problem. Hence, an optimal solution for the k-cMEC problem is not necessarily an optimal solution for the MEC problem.

2.3 Algorithm

In this section, we present an FPT algorithm for solving the k-cMEC problem, when parameterized by the maximum number k of corrections that are allowed in each column and by the coverage cov. The algorithm is based on an exact dynamic programming approach. After presenting the basic dynamic programming equation for the gapless case, we show that the approach can be easily adapted to manage gaps and, possibly, the all-heterozygous assumption.

Informally, the algorithm iteratively computes, for all j from 1 to m, a k-corrected matrix $\mathcal{M}'$ on the first j columns $\mathcal{M}_1, \ldots, \mathcal{M}_j$ of the input matrix $\mathcal{M}$ by considering all the possible corrections $\mathcal{M}'_j$ for the last column $\mathcal{M}_j$ such that $d(\mathcal{M}_j, \mathcal{M}'_j) \leq k$ and choosing the best option. The corrected column $\mathcal{M}'_j$ can be either homozygous or heterozygous. If it is homozygous, we pay a cost equal to the homozygous distance $H(\mathcal{M}_j)$ of $\mathcal{M}_j$ but then $\mathcal{M}'_j$ is in accordance with any other column and no other check must be performed. If $\mathcal{M}'_j$ is heterozygous, then we consider all the possible k-corrections B_j for $\mathcal{M}_j$ in β_j and for each one two different cases may arise: (i) there exists a column $\mathcal{M}_q$ with $q < j$ that 'shares' some fragments with $\mathcal{M}_j$ and that in the optimal solution $\mathcal{M}'_q$ is heterozygous (clearly, $q \geq j - L$) or (ii) all the previous columns that share some fragments with $\mathcal{M}_j$ are homozygous in the optimal solution $\mathcal{M}'$. In the first case, $\mathcal{M}'_q$ and $\mathcal{M}'_j$ must be either identical or complementary on the shared fragments (Lemma 1). It follows that we have to choose the best option among all the k-corrections B_q such that $d(B_q, B_j) = 0$ and we pay a cost equal to that of the correction B_j (i.e. $d(\mathcal{M}_j, B_j)$) plus the cost of transforming the columns between $\mathcal{M}_q$ and $\mathcal{M}_j$ into homozygous columns (i.e. their homozygous distance). In the second case, all the columns to the left of $\mathcal{M}_j$ that share some fragment with $\mathcal{M}_j$ are homozygous in the optimal solution $\mathcal{M}'$ (and we pay a total cost equal to their homozygous distance). As a consequence, any k-correction B_j of $\mathcal{M}_j$ is in accordance with them and we pay a cost equal to $d(\mathcal{M}_j, B_j)$.

More formally, let $\mathcal{M}$ be a fragment matrix and B_j be a k-correction for $\mathcal{M}_j$, we define $D[j, B_j]$ as the minimum number of corrections needed to obtain a k-corrected matrix $\mathcal{M}'$ for $\mathcal{M}$ on columns $\mathcal{M}_1, \ldots, \mathcal{M}_j$ such that $\mathcal{M}'_j$ is heterozygous and $d(\mathcal{M}'_j, B_j) = 0$. Moreover, we define OPT$[j]$ as the minimum number of corrections needed to obtain a k-corrected $\mathcal{M}'$ for $\mathcal{M}$ on columns $\mathcal{M}_1, \ldots, \mathcal{M}_j$. Finally, we define $\mathcal{M}_{L_j}$ ($\mathcal{M}_{R_j}$, respectively) as the rightmost (leftmost, respectively) column to the left (right, respectively) of $\mathcal{M}_j$ such that active$(\mathcal{M}_{L_j}, \mathcal{M}_j) = \phi$ (active$(\mathcal{M}_{R_j}, \mathcal{M}_j) = \phi$, respectively); if it does not exist, $\mathcal{M}_{L_j}$ ($\mathcal{M}_{R_j}$, respectively) corresponds to an empty column in position 0 ($m+1$, respectively). Note that $j - L_j \leq L$ and $R_j - j \leq L$.

Without loss of generality, we implicitly assume that there exists a dummy empty column $\mathcal{M}_0$ in position 0 of the input $\mathcal{M}$. Thus, we can define OPT$[0] = 0$ and $D[0, \cdot] = 0$.

For $0 < j \leq m$, $D[j, B_j]$ and OPT$[j]$ can be computed as follows:

$$D[j, B_j] = \min \begin{cases} \min\limits_{\substack{q\,:\,L_j+1 \leq q \leq j-1,\\ B_q\,:\,d(B_j,B_q)=0}} D[q, B_q] + d(\mathcal{M}_j, B_j) + \sum\limits_{y=q+1}^{j-1} H(\mathcal{M}_y) \\ OPT[L_j] + d(\mathcal{M}_j, B_j) + \sum\limits_{y=L_j+1}^{j-1} H(\mathcal{M}_y) \end{cases} \tag{1}$$

$$\text{OPT}[j] = \min \begin{cases} \text{OPT}[j-1] + H(\mathcal{M}_j) & //\mathcal{M}_j \text{ is homozygous} \\ \min_{\forall B_j} D[j, B_j] & //\mathcal{M}_j \text{ is heterozygous} \end{cases} \tag{2}$$

The optimum cost is given by OPT$[m]$ and a corresponding optimal solution $\mathcal{M}'$ can be reconstructed by backtracking. The formal proof of correctness along with some technical details about the backtracking procedure are in the Supplementary Material.

The algorithm can be easily adapted to the weighted version of k-cMEC. In this case, each non-hole element $\mathcal{M}_j[i]$ of the input matrix $\mathcal{M}$ has a weight $w(\mathcal{M}_j[i])$. Given a column $\mathcal{M}_j$ and any k-correction B_j in β_j, the key idea is to consider the weight $w(\mathcal{M}_j, B_j)$ as the minimum sum of the weights to transform $\mathcal{M}_j$ in $\mathcal{M}'_j$ such that $d(\mathcal{M}'_j, B_j) = 0$ and to consider the weight $w_H(\mathcal{M}_j)$ as the minimum sum of weights to transform $\mathcal{M}_j$ into a homozygous column. Hence, we want to minimize the sum of such weights by replacing $d(\mathcal{M}_j, B_j)$ with $w(\mathcal{M}_j, B_j)$ and $H(\mathcal{M}_j)$ with $w_H(\mathcal{M}_j)$ in the recursive equations.

Assume to consider a general fragment matrix $\mathcal{M}$ that may contain gaps. As explained before, any gap can be modeled as zero-weight elements. Since each of these elements can be equal to 0 or 1 with a cost of 0, we can adapt the algorithm such that for each column all the combinations of values for its gaps will be considered. It follows that any k-correction B_j for a column $\mathcal{M}_j$ is extended with any combination of values for its gaps and added to β_j.

Furthermore, the algorithm can be slightly modified to find a solution under the all-heterozygous assumption that forces to reconstruct two complementary haplotypes. In this case, the homozygous columns, both in the input and in the output, have to be considered as 'special' heterozygous columns that place all the covered fragments in the same part of the fragments bipartition. Hence, we remove from the recursive equation the possibility to transform each column $\mathcal{M}_j$ into a homozygous column and we add to β_j the k-correction B_j that transforms $\mathcal{M}_j$ into a 'special' heterozygous column.

We now show how the time complexity $O((\sum_{s=0}^{k} \binom{\text{cov}}{s})^2 \cdot \text{cov} \cdot L \cdot m)$ of a naive implementation of the recursive equation can be reduced to $O(\sum_{s=0}^{k} \binom{\text{cov}}{s} \cdot \text{cov} \cdot L \cdot m)$ by a careful re-engineering of the algorithm.

First, given n elements, their (n, s)-combinations for all s between 0 and k are $\sum_{s=0}^{k} \binom{n}{s}$ and they correspond to all the k-corrections B_j in β_j. Such a set can be enumerated in lexicographic order in time $O(\sum_{s=0}^{k} \binom{\text{cov}}{s})$ (Knuth, 2005, see Alg. T, chapter 7.2.1.3) and, using the same ideas, it is possible to compute the ith element (for any arbitrary i) of this order in time $O(k)$.

In a direct implementation of the recurrence equations, there exists $O(m \cdot \sum_{s=0}^{k} \binom{\text{cov}}{s})$ entries $D[j, B_j]$ and m entries OPT$[j]$. Computing each entry $D[j, B_j]$ requires checking at most one entry OPT$[q]$ and checking at most $L \cdot \sum_{s=0}^{k} \binom{\text{cov}}{s}$ entries $D[q, B_q]$ (for any q in $\{L_j, \ldots, j-1\}$) in time O(cov) each (for computing $d(B_j, B_q)$, since $\text{cov}_j, \text{cov}_q \leq \text{cov}$). Therefore, since OPT$[j]$ can be updated in constant time during the computation of the entries

$D[j, B_j]$, we have that the overall time complexity of the simple implementation is $O((\sum_{s=0}^{k}\binom{\text{cov}}{s})^2 \cdot \text{cov} \cdot L \cdot m)$.

The time complexity can be improved to $O(\sum_{s=0}^{k}\binom{\text{cov}}{s} \cdot \text{cov} \cdot L \cdot m)$ by computing an *intermediate projection table* and applying an approach inspired by the one presented in Patterson *et al.* (2014). Let $\mathcal{M}$ be a gapless fragment matrix. Given two columns $\mathcal{M}_{j_1}$ and $\mathcal{M}_{j_2}$, and a k-correction B_{j_1} for $\mathcal{M}_{j_1}$, we define $\pi_{j_2}(B_{j_1})$ as the vector of size $|\text{active}(\mathcal{M}_{j_1}, \mathcal{M}_{j_2})|$ that is obtained from B_{j_1} by keeping only elements that correspond to fragments that are in $\text{active}(\mathcal{M}_{j_1}, \mathcal{M}_{j_2})$. We define the intermediate projection table for each column $\mathcal{M}_j$, for each q in $\{L_j + 1, \ldots, j-1\}$ and for each vector C representing a possible correction of the positions in $\text{active}(\mathcal{M}_j, \mathcal{M}_q)$, as follows:

$$\tilde{D}[q, j, C] = \min_{\forall B_q \in \beta_q : d(C, \pi_j(B_q)) = 0} D[q, B_q]. \quad (3)$$

Entry $\tilde{D}[q, j, \pi_j(B_q)]$ (and $\tilde{D}[q, j, \overline{\pi_j(B_q)}]$, where $\overline{\pi_j(B_q)}$ is the complement of $\pi_j(B_q)$) can be filled in $O(\text{cov})$ time [needed to compute $\pi_j(B_q)$] while computing $D[q, B_q]$ and, consequently, the asymptotic overall time complexity does not change. Intuitively, $\tilde{D}[q, j, C]$ corresponds to the minimum number of corrections to obtain a k-corrected matrix $\mathcal{M}'$ for $\mathcal{M}$ on the first q columns such that $\mathcal{M}'_q$ is heterozygous and $d(C, \pi_j(\mathcal{M}'_q)) = 0$. As a consequence, Equation (1) can be equivalently rewritten as:

$$D[j, B_j] = \min \begin{cases} \tilde{D}[q, j, \pi_q(B_j)] + d(\mathcal{M}_j, B_j) + \sum_{y=q+1}^{j-1} H(\mathcal{M}_y) \\ \text{OPT}[L_j] + d(\mathcal{M}_j, B_j) + \sum_{y=L_j+1}^{j-1} H(\mathcal{M}_y) \end{cases} \quad (4)$$

In other words, with this recurrence, each entry $D[j, B_j]$ is computed using the entries $\tilde{D}[\cdot, j, \cdot]$ and, at the same time, it is used to update the entries $\tilde{D}[j, \cdot, \cdot]$ and $OPT[j]$ without changing the asymptotic time complexity. Since there exists $O(m \cdot L \cdot \sum_{s=0}^{k}\binom{\text{cov}}{s})$ entries $\tilde{D}[j, p, \pi_p(B_j)]$ with p in $\{j+1, \ldots, L-1\}$, it follows that the overall time complexity is $O(\sum_{s=0}^{k}\binom{\text{cov}}{s} \cdot \text{cov} \cdot L \cdot m)$. Notice that $O(\text{cov}^{k+1} \cdot L \cdot m)$ is a more intuitive, but less tight, bound.

Concerning space complexity and according to Equation (4), the intermediate projection table $\tilde{D}[\cdot, \cdot, \cdot]$ can be stored instead of table $D[\cdot, \cdot]$. This takes $O(L \cdot m \cdot \sum_{s=0}^{k}\binom{\text{cov}}{s})$ space, since for any column $\mathcal{M}_j$ we only consider all the values q in $\{L_j + 1, \ldots, j-1\}$ and $j - L_j \leq L$. Therefore, since the algorithm iteratively proceeds column wise, when it is at the step corresponding to the column $\mathcal{M}_j$, we just need to consider the entries $\tilde{D}[y, \cdot, \cdot]$ for all the columns $\mathcal{M}_y$ with $j \leq y \leq R_j$. For this reason, we just need $O(L \cdot L \cdot \sum_{s=0}^{k}\binom{\text{cov}}{s})$ space to store that window of the projection table.

Furthermore, if we consider a general fragment matrix $\mathcal{M}$ modelling the gaps as zero-weight elements (using the approach described before), the number of k-corrections B_j in β_j for a column $\mathcal{M}_j$ increases to $2^g \cdot \sum_{s=0}^{k}\binom{\text{cov}}{s}$, where g is the maximum number of gaps in a column (hence 2^g is the number of all the combinations of values for gap elements). As a consequence, the overall time complexity becomes $O(2^g \cdot \sum_{s=0}^{k}\binom{\text{cov}}{s} \cdot \text{cov} \cdot L \cdot m)$ and it takes $O(2^g \cdot L \cdot L \cdot \sum_{s=0}^{k}\binom{\text{cov}}{s})$ space.

As shown in the Supplementary Material, for the backtracking phase we need two tables requiring $O(m)$ and $O(m \cdot \sum_{s=0}^{k}\binom{\text{cov}}{s})$ space, respectively.

2.4 Implementation

A prototypical implementation of HapCol is available under the terms of the GPL at http://hapcol.algolab.eu/. Since coverage varies across columns, HapCol adaptively adopts a different maximum number k_j of corrections for each column $\mathcal{M}_j$ computed as the smallest integer such that the probability that $\mathcal{M}_j$ contains more than k_j errors is at most α, with α given as input. Such a probability is computed assuming that sequencing errors are uniformly distributed with a substitution error rate ϵ (given as input), an assumption which reflects the characteristics of future-generation sequencing technologies. Therefore, the two parameters given in input to HapCol are ϵ and α and can be chosen by the user depending on the estimated sequencing (substitution) error rate and on the user's preference towards better performances (larger α) or increased probability of finding a feasible solution (smaller α).

The strategy currently implemented for choosing the maximum number of corrections per column assumes that errors are uniformly distributed. However, it can be easily modified to process datasets produced by technologies with different error profiles (even those with systematic errors, especially if the average error rate is low, such as current Illumina technologies) and/or to automatically increase the values k_j until a feasible solution exists.

3 Results and discussion

We experimentally compared accuracy and performance of HapCol with those of state-of-the-art haplotype assembly approaches on both real (Section 3.1) and simulated datasets (Section 3.2). The experimental comparison on the real long read dataset is focused on evaluating the accuracy of the tools under the *all-heterozygous* assumption since such a standard benchmark dataset has low average coverage (~3×) and contains only heterozygous SNP positions. We also assessed accuracy and performances of the tools while varying coverage, read length and sequencing/indel error rate on simulated long read datasets with characteristics similar to those of the 'future-generation' sequencing technologies that are currently (or soon) available (coverage up to 25×, read length up to 50 000 bases, substitution error rate up to 5% and indel rate equal to 10%) (Carneiro *et al.*, 2012; Jain *et al.*, 2015; Roberts *et al.*, 2013).

We compared HapCol with three state-of-the-art haplotyping tools specifically designed for handling long reads, namely, RefHap, which was shown to be one of the most accurate heuristic methods (Duitama *et al.*, 2012), ProbHap, a recent probabilistic method which has been shown to be sensibly more accurate than RefHap (Kuleshov, 2014) and WhatsHap, the first exact approach for the weighted MEC problem specifically designed for long reads (Patterson *et al.*, 2014, 2015). At higher coverages, applications such as SNP calling or validating which SNPs are really heterozygous in the given sample (e.g. there could be a significant portion of positions that, due to sequencing errors, appears to be heterozygous, but that should be predicted as homozygous) become feasible and reliable (Nielsen *et al.*, 2011). However, since these applications require that haplotypes are reconstructed without the all-heterozygous assumption, on the simulated datasets we only considered WhatsHap and HapCol as they do not rely on this assumption.

The analyses focused on the accuracy of the reconstructed haplotypes and on the performances of the tools. Accuracy of the reconstructed haplotypes has been evaluated in terms of *(switch) error rate* (Browning and Browning, 2011) (i.e. the number of inconsistencies over contiguous phased variants) and in terms of phased positions (i.e. the number of positions for which the tool gave a phase prediction over the total number of positions that can be phased using the fragments given as input). Performances of the tools have been evaluated in terms of running time and peak memory usage, as reported by the Unix utility `time`. All the tests have been performed on a server equipped with four Intel Xeon E5-4610v2 CPUs and 256 GB of RAM.

3.1 The NA12878 dataset

The real dataset (called 'NA12878 dataset') is the one produced using a fosmid-based technology from the HapMap sample NA12878 by Duitama *et al.* (2012). This dataset is considered a standard benchmark for comparing haplotyping algorithms on long reads, since the haplotypes of individual NA12878 were independently and confidently reconstructed using the sequenced genomes of the individual and of her parents. The dataset is composed of 271 184 reads with average length of ∼40 kb and with average coverage of ∼3×. The reference haplotypes are the trio-phased variant calls from the GATK resource bundle (DePristo *et al.*, 2011), filtered on the 1 252 769 positions that are also covered by the fragments of the NA12878 dataset.

HapCol, ReFHap, ProbHap and WhatsHap have been executed independently on each chromosome. HapCol and WhatsHap can be executed with or without the all-heterozygous assumption without affecting the exponential part of their time/space complexities. In this case, these two tools have been executed using the all-heterozygous assumption, since the positions covered by the dataset are heterozygous with high confidence and since the comparison between solutions obtained with different assumptions may lead to misleading results. Moreover, HapCol has been executed with $\epsilon = 5\%$ and $\alpha = 10^{-3}$ and, for this choice of the parameters, a feasible solution existed for each chromosome. Table 1 reports, for each tool, the overall error rate and the percentage of phased positions over all the phasable positions, the total running time and the peak of memory for the whole dataset (i.e. for all the chromosomes).

On this dataset, HapCol reconstructed the most accurate haplotypes and phased the largest number of positions compared with the other tools. In particular, HapCol improves the accuracy obtained by WhatsHap, ProbHap and ReFHap by around 6%, 43% and 48%, respectively. Furthermore, HapCol is also the tool which phases the largest number of positions. In fact, HapCol phases 0.15% more positions than WhatsHap, 2.03% more than ProbHap and 2.18% more than ReFHap.

To the contrary, ReFHap was the fastest and most memory-efficient tool among the four considered. This was expected, as ReFHap is a heuristic-based method, while the other ones are exact (albeit they minimize different objective functions). Overall, all the tools can be run with modest/medium computing resources. Indeed, each one analyzed the dataset in less than 25 min and using less than 24 GB of memory. However, while HapCol and WhatsHap concluded in a few minutes, ProbHap was significantly slower than the others (∼20 min) and, possibly, it could not be able to scale to datasets with higher coverage.

HapCol and WhatsHap required significantly more memory than ProbHap and ReFHap (4 times and 44 times, respectively). However, such a peak of memory usage is due to a small number of consecutive positions on chromosomes 2, 3 and 10 where coverage is high (up to 30×), but most of values are gaps (all but 2-4 non-gap alleles on average). In these regions, the performances of HapCol and WhatsHap degrade since both approaches model the gaps as zero-weight elements and must essentially 'guess' the alleles at those positions. Clearly, in this case, phase prediction is not reliable and a simple pre-filtering step can easily find (and possibly remove) such positions from further analyses. If we exclude chromosomes 2, 3 and 10, then WhatsHap becomes the fastest tool (30 s), followed by ReFHap (35 s), by HapCol (60 s) and by ProbHap that remains the slowest tool (956 s). In terms of memory usage, HapCol turns out to be the most memory-efficient method (0.06 GB), followed by WhatsHap (0.16 GB), by ProbHap (0.48 GB) and by ReFHap (0.54 GB).

Table 1. Comparison of four haplotyping tools on the NA12878 real dataset

Tool	Error (%)	Phased (%)	Time (s)	Mem. (GB)
HapCol	**1.91**	**99.88**	332	2.1
WhatsHap	2.02	99.73	172	23.9
ProbHap	3.36	98.02	1205	0.6
ReFHap	3.68	97.75	**43**	**0.5**

Accuracy is given in terms of phasing error ('error') and total phased positions ('phased') of the reconstructed haplotypes, while performances are given in terms of total running time ('time') expressed in seconds and the peak memory usage ('mem.') expressed in GB. Best results for each column are highlighted in boldface.

3.2 Simulated datasets

We used simulated datasets to assess how accuracy and performances change while the characteristics of the dataset (coverage, especially) vary. As motivated before, in this part we focused on the tools that can work also without the all-heterozygous assumption, namely HapCol and WhatsHap. The simulation of the datasets has been performed as in Patterson *et al.* (2015). The dataset consists of a ground truth, which was assembled by inserting all known variants of chromosomes 1 and 15 of J. Craig Venter's genome into the

Table 2. Comparison of HapCol (hc) and WhatsHap (wh) on realistically simulated instances.

			Chromosome 15							Chromosome 1						
			Feas.	Error (%)		Time (s)		Mem. (GB)		Feas.	Error (%)		Time (s)		Mem. (GB)	
ϵ / α	cov	e	·/20	wh	hc	wh	hc	wh	hc	·/20	wh	hc	wh	hc	wh	hc
5% / 10^{-2}	15×	1	17	2.26	2.24	18	6	1.7	0.1	15	2.40	2.40	47	17	4.5	0.3
		5	20	1.98	1.98	19	6	1.8	0.1	8	2.42	2.44	46	17	4.4	0.3
	20×	1	18	1.77	1.76	487	53	52.9	0.6	7	1.84	1.84	1241	155	129.2	2.0
		5	18	1.76	1.76	490	48	52.7	0.6	4	2.07	2.08	1249	132	129.0	1.6
5% / 10^{-3}	15×	1	20	2.12	2.11	19	25	1.8	0.3	20	2.35	2.36	48	64	4.6	0.8
		5	20	1.98	1.98	19	22	1.8	0.3	19	2.35	2.35	49	56	4.7	0.7
	20×	1	20	1.82	1.81	485	218	52.8	2.2	19	1.95	1.94	1306	586	138.0	5.6
		5	20	1.67	1.67	497	200	53.6	2.0	19	2.07	2.08	1347	526	138.5	5.1

The simulated instances have coverage ('cov') 15× and 20×, substitution error rate ('e') 1% and 5% and indel error rate fixed to 10%. The metrics considered are the number of instances with feasible solutions ('feas.'), the average phasing error ('error') of the reconstructed haplotypes, the average running time ('time') and the average maximum used memory ('mem.'). HapCol has been executed with two different combinations of ϵ and α: 5% / 10^{-2} and 5% / 10^{-3}.

human reference genome (hg18), mapped simulated long reads of lengths 1000, 5000, 10 000 and 50 000 bases with varying uniform substitution rates 1% and 5% and with a uniform indel distribution of 10% at 30× coverage. These rates reflect the characteristics of the long read data generated by the future-generation sequencing technologies (Carneiro *et al.*, 2012; Jain *et al.*, 2015; Roberts *et al.*, 2013). From each set of simulated reads, five datasets were obtained by randomly extracting a maximal subset with (maximum) coverage of 15×, 20× and 25×.

HapCol has been executed with two combinations of its input parameters—namely $\epsilon = 5\%$ with $\alpha = 10^{-2}$, and $\epsilon = 5\%$ with $\alpha = 10^{-3}$—to assess the behavior of HapCol depending on the choice of the parameters. We remark that some fragment matrices could not admit a feasible solution for the *k*-cMEC problem with some choices of parameter *k* (depending on ϵ and α in the implementation), while the same instances have always a feasible solution for the (unconstrained) MEC problem. Both tools have been executed on all the instances, but HapCol terminated on some of them because no feasible solution existed for that choice of the input parameters. WhatsHap, which should be able to find a feasible solution for all the instances, computed a solution only for the instances with coverage 15× and 20×, while, as expected (Patterson *et al.*, 2015), it was not able to successfully conclude the execution on the instances with coverage 25× since it exhausted the available memory (256 GB).

Table 2 reports, for any combination of input parameters ϵ and α, the number of instances with a feasible solution (column 'feas.'), the average error of the reconstructed haplotypes, the average running time and the average memory usage over all the instances of a given chromosome (Venter chromosome 1 and chromosome 15), coverage (15× and 20×) and substitution error rate *e* (1% and 5%) (the indel error rate is fixed to 10%, thus is not reported). The results presented in the table refer only to the subset of instances which have a feasible solution for the *k*-cMEC problem and are averaged over the read lengths. Since WhatsHap was not able to successfully terminate on any instance with coverage 25×, the results on that subset of instances are separately reported (only for HapCol) on Table 3.

First, as expected, the number of instances with a feasible solution increases as the combination of parameters ϵ and α allows more corrections per column. Indeed, for $\epsilon = 5\%$ and $\alpha = 10^{-2}$, the maximum numbers of corrections per column (not shown) are quite low and, as a consequence, a feasible solution does not exists for many instances, especially for those with high substitution error rates *e*. For the other combination of parameters (namely, $\epsilon = 5\%$ and $\alpha = 10^{-3}$), the number of instances with a feasible solution rapidly increases. This trend, albeit less evident for chromosome 15, is clear for chromosome 1. Noticeably, when $\epsilon = 5\%$ and $\alpha = 10^{-3}$, a feasible solution exists for all but three instances with coverage at most 20×.

In terms of accuracy of the reconstructed haplotypes, on all the instances, HapCol obtained the same phasing error rate of WhatsHap, which, in turn, was shown to be competitive with other state-of-the-art approaches (Patterson *et al.*, 2015). This observation supports the validity of the newly introduced *k*-cMEC problem as a computational model for haplotype assembly on long reads.

Albeit HapCol and WhatsHap achieve the same accuracy, in terms of performances HapCol is both faster and significantly more memory-efficient than WhatsHap. In particular, on average, HapCol is at least twice faster than WhatsHap when the coverage is 20× even for the largest values of maximum number k_j of corrections per column (i.e. when $\epsilon = 5\%$ and $\alpha = 10^{-3}$). Moreover, with the same parameters and with read length of 10 000 bases (a typical average read length in the foreseeable future), HapCol is almost three times faster than WhatsHap, allowing to process a single dataset in less than 11 min (on average). Concerning memory usage, we observe the same general trend, except that differences are even more evident. In fact, the average memory usage of WhatsHap on chromosome 1 (the largest one) at coverage 20× is ∼138 GB, while HapCol requires only ∼5 GB. Moreover, on instances with read length at most 50 000 bases, WhatsHap requires up to 164 GB, while HapCol never requires more than 10 GB. As a consequence, with HapCol, the analysis of a genome-wide dataset at coverage 20× is feasible even on a standard workstation/small server.

As noticed before, three instances do not admit a feasible solution with $\epsilon = 5\%$ and $\alpha = 10^{-3}$. However, by setting $\epsilon = 5\%$ and $\alpha = 10^{-4}$, a feasible solution exists also for these instances and the error rate of the solution obtained by HapCol is equal to that obtained by WhatsHap. In terms of performance, HapCol is slower than WhatsHap on the single instance with coverage 15× and it has a similar running time of WhatsHap on the two instances with coverage 20× (∼21 min, on average). Noticeably, the amount of memory required by HapCol on these two instances (∼9 GB, on average) is ∼15 times lower than that required by WhatsHap (∼143 GB, on average), a further confirmation of the memory-efficiency of HapCol even when more corrections per columns are allowed. Furthermore, these results confirm that a simple strategy that progressively increases the number of corrections allowed in each column until a solution is found would be practicable, since it always leads to a solution while keeping the memory usage as low as possible.

A comparison between HapCol and WhatsHap is not possible on instances with coverage 25×, since WhatsHap was not able to solve these instances within the available amount of memory. Hence, we evaluated how accuracy and performances of HapCol vary between instances with coverage 20× and 25× (Table 3). For space constraints, we report the results only for $\epsilon = 5\%$ and $\alpha = 10^{-3}$, that is for the parameters for which the maximum number of instances has a feasible solution. Moreover, to not discard the effect of read lengths, we focused only on instances with read length

Table 3. Comparison between instances with coverage 20x and 25× for HapCol.

		Chromosome 15				Chromosome 1			
e	cov	Feas. ·/5	Error (%)	Time (s)	Mem. (GB)	Feas. ·/5	Error (%)	Time (s)	Mem. (GB)
1	20×	5	1.40	311	3.8	5	1.66	832	9.5
	25×	5	1.25	1457	16.1	4	1.52	4272	40.7
5	20×	5	1.24	277	3.3	5	1.71	737	8.5
	25×	5	1.14	1466	15.2	4	1.55	4357	39.2

HapCol has been executed with $\epsilon = 5\%$ and $\alpha = 10^{-3}$. Read length was fixed to 50 000 bases and the attributes are the same of Table 2.

50 000 bases. Such a read length has been chosen since almost all these instances have a feasible solution for both coverage 20× and 25×. We observe that increasing coverage from 20× to 25× allows to improve accuracy (∼9%) of the reconstructed haplotypes (as we already observed for coverage 15× and 20×). Moreover, the increased accuracy is mainly due to a significant reduction (approximately −10% on average, data not shown for space constraints) of the number of ambiguous positions, leading to an increased number of phased SNP positions. Concerning performances, instances with coverage 25× can be still analyzed with modest computing equipments; indeed HapCol completed the tests on chromosome 1 in <73 min and using less than 40 GB of main memory.

4 Conclusion

We have proposed an exact algorithm, called HapCol, for the weighted *k*-cMEC, a new variant of the wMEC problem that takes into account the main characteristics of future-generation sequencing technologies, namely the uniform distribution of sequencing errors and the increasing length of sequenced reads.

We showed that the haplotypes computed by HapCol on a real benchmark dataset are at least as accurate as those computed by current state-of-the-art approaches. This result supports the validity of the additional constraints imposed by the *k*-cMEC problem.

Furthermore, HapCol is able to overcome the traditional all-heterozygous assumption and to process datasets with coverage 25× on standard workstations/small servers, while the current state-of-the-art methods either rely on this assumption or become unfeasible on coverages over 20×. Thanks to these results, HapCol is potentially able to directly perform SNP calling or heterozygous SNPs validation that become feasible and reliable on coverage up to 25×.

HapCol has been specifically designed to exploit the uniform distribution of errors that characterizes 'future-generation' sequencing technologies, but it can be successfully applied on sequencing data with a different error distribution by choosing the maximum number *k* of errors per position according to the error model. Furthermore, HapCol can be easily extended to adaptively increase the value of *k* (on certain columns) until a feasible solution exists. This strategy allows to process datasets affected by systematic sequencing errors without a great impact on the performance if the average error rate is low (such as in the current Illumina sequencing technologies).

An interesting future direction would be the extension of the *k*-cMEC problem to deal with individuals related by structures such as trios or pedigrees (Browning and Browning, 2011). Indeed, the Mendelian laws of inheritance induce further constraints that may improve the accuracy of the reconstructed haplotypes, as shown for example by Pirola *et al.* (2012).

Acknowledgements

We thank Tobias Marschall for his helpful suggestions on the data generation and evaluation analysis. This work was stimulated by discussions between P.B., G.W.K. and N.P. during the No. 045 NII Shonan workshop on Exact Algorithms for Bioinformatics Research, March 2014, Japan.

Funding

This work was partially supported by MIUR [2010LYA9RH to P.B., R.D., Y.P. and S.Z.]; by Fondazione Cariplo [2013-0955 to P.B., Y.P. and S.Z.] and by Univ. degli Studi di Milano-Bicocca [FA 2013 grant to P.B., Y.P. and S.Z.].

Conflict of Interest: none declared.

References

Aguiar,D. and Istrail,S. (2012) HapCompass: a fast cycle basis algorithm for accurate haplotype assembly of sequence data. *J. Comput. Biol.,* **19**, 577–590.

Bansal,V. and Bafna,V. (2008) HapCUT: an efficient and accurate algorithm for the haplotype assembly problem. *Bioinformatics,* **24**, i153–i159.

Bonizzoni,P. *et al.* (2015) On the fixed parameter tractability and approximability of the minimum error correction problem. In: Ferdinando,C. *et al.* (eds) *CPM,* volume 9133 of *LNCS*, Springer International Publishing, pp. 100–113.

Browning,S. and Browning,B. (2011) Haplotype phasing: existing methods and new developments. *Nat. Rev. Genet.,* **12**, 703–714.

Carneiro,M. *et al.* (2012) Pacific biosciences sequencing technology for genotyping and variation discovery in human data. *BMC Genomics,* **13**, 375.

Chen,Z.-Z. *et al.* (2013) Exact algorithms for haplotype assembly from whole-genome sequence data. *Bioinformatics,* **29**, 1938–1945.

Cilibrasi,R. *et al.* (2007) The complexity of the single individual SNP haplotyping problem. *Algorithmica,* **49**, 13–36.

DePristo,M. *et al.* (2011) A framework for variation discovery and genotyping using next-generation DNA sequencing data. *Nat. Genet.,* **43**, 491–498.

Duitama,J. *et al.* (2010) ReFHap: a reliable and fast algorithm for single individual haplotyping. In: Li,L. *et al.* (eds) *BCB*. ACM, New York, NY, USA, pp. 160–169.

Duitama,J. *et al.* (2012) Fosmid-based whole genome haplotyping of a HapMap trio child: evaluation of single individual haplotyping techniques. *Nucleic Acids Res.,* **40**, 2041–2053.

Greenberg,H. *et al.* (2004) Opportunities for combinatorial optimization in computational biology. *INFORMS J. Comput.,* **16**, 211–231.

He,D. *et al.* (2010) Optimal algorithms for haplotype assembly from whole-genome sequence data. *Bioinformatics,* **26**, i183–i190.

He,D. *et al.* (2013) Hap-seq: an optimal algorithm for haplotype phasing with imputation using sequencing data. *J. Comput. Biol.,* **20**, 80–92.

Jain,M. *et al.* (2015) Improved data analysis for the minion nanopore sequencer. *Nat. Methods,* **12**, 351–356.

Knuth,D.E. (2005) *The Art of Computer Programming,* Vol. 4. Addison-Wesley, New York City, New York.

Kuleshov,V. (2014) Probabilistic single-individual haplotyping. *Bioinformatics,* **30**, i379–i385.

Kuleshov,V. *et al.* (2014) Whole-genome haplotyping using long reads and statistical methods. *Nat. Biotechnol.,* **32**, 261, 266.

Lancia,G. *et al.* (2001) SNPs problems, complexity, and algorithms. In: auf der Heide,F.M. (ed) *ESA,* volume 2161 of *LNCS,* Springer Berlin Heidelberg, pp. 182–193.

Lippert,R. *et al.* (2002) Algorithmic strategies for the single nucleotide polymorphism haplotype assembly problem. *Brief. Bioinform.,* **3**, 23–31.

Nielsen,R. *et al.* (2011) Genotype and SNP calling from next-generation sequencing data. *Nat. Rev. Genet.,* **12**, 443–451.

Patterson,M. *et al.* (2014) WhatsHap: haplotype assembly for future-generation sequencing reads. In: Roded,S. (ed) *RECOMB,* volume 8394 of *LNCS,* Springer International Publishing, pp. 237–249.

Patterson,M. *et al.* (2015) WhatsHap: weighted haplotype assembly for future-generation sequencing reads. *J. Comput. Biol.,* **6**, 498–509.

Pirola,Y. *et al.* (2012) An efficient algorithm for haplotype inference on pedigrees with recombinations and mutations. *IEEE/ACM Trans. Comput. Biol. Bioinform.*, **9**, 12–25.

Roberts,R.J. *et al.* (2013) The advantages of SMRT sequencing. *Genome Biol.,* **14**, 405.

Smith,C.C. *et al.* (2012) Validation of ITD mutations in FLT3 as a therapeutic target in human acute myeloid leukaemia. *Nature,* **485**, 260–263.

Zhao,Y.-Y. *et al.* (2005) Haplotype assembly from aligned weighted SNP fragments. *Comput. Biol. Chem.,* **29**, 281–287.