HapCompass: A fast cycle basis algorithm for accurate haplotype assembly of sequence data

Derek Aguiar [1,2] Sorin Istrail [1,2,*]

Dedicated to professors Michael Waterman's 70th birthday and Simon Tavare's 60th birthday

[1]Department of Computer Science, Brown University, Providence, RI, USA

[2]Center for Computational Molecular Biology, Brown University, Providence, RI, USA

Abstract: Genome assembly methods produce haplotype phase ambiguous assemblies due to limitations in current sequencing technologies. Determining the haplotype phase of an individual is computationally challenging and experimentally expensive. However, haplotype phase information is crucial in many bioinformatics workflows such as genetic association studies and genomic imputation. Current computational methods of determining haplotype phase from sequence data – known as haplotype assembly – have

[*]to whom correspondence should be addressed: Sorin_Istrail@brown.edu

difficulties producing accurate results for large (1000 genomes-type) data or operate on restricted optimizations that are unrealistic considering modern high-throughput sequencing technologies.

We present a novel algorithm, HapCompass, for haplotype assembly of densely sequenced human genome data. The HapCompass algorithm operates on a graph where single nucleotide polymorphisms (SNPs) are nodes and edges are defined by sequence reads and viewed as supporting evidence of co-occuring SNP alleles in a haplotype. In our graph model, haplotype phasings correspond to spanning trees and each spanning tree uniquely defines a cycle basis. We define the minimum weighted edge removal global optimization on this graph and develop an algorithm based on local optimizations of the cycle basis for resolving conflicting evidence. We then estimate the amount of sequencing required to produce a complete haplotype assembly of a chromosome. Using these estimates together with metrics borrowed from genome assembly and haplotype phasing, we compare the accuracy of HapCompass, the Genome Analysis ToolKit, and HapCut for 1000 genomes and simulated data. We show that HapCompass performs significantly better for a variety of data and metrics. HapCompass is freely available for download at http://www.brown.edu/Research/Istrail_Lab/.

1 Introduction

High-throughput DNA sequencing technologies are producing increasingly abundant and long sequence reads. Third generation technologies promise to output even longer reads (up to a few kb) with increasingly long insert sizes. While the latter promises to alleviate many of the difficulties associated with high-throughput sequencing pipelines, both technologies suffer from producing haplotype phase ambiguous sequence reads. Determining the haplotype phase of an individual is computationally challenging and experimentally expensive; but haplotype phase information is crucial in bioinformatics workflows (Tewhey et al. [2011]) including genetic association studies and identifying components of the missing heritability problem (e.g., phase-dependent interactions like compound heterozygosity (Krawitz et al., 2010, Pierson et al., 2012)), the reconstruction of phylogenies and pedigrees, genomic imputation (Marchini and Howie, 2010), linkage disequilibrium and SNP tagging (Tarpine et al., 2011).

Two categories of computational methods exist for determining haplotypes: haplotype phasing and haplotype assembly. Given the genotypes of a sample of individuals from a population, *haplotype phasing* attempts to infer the haplotypes of the sample using haplotype sharing information within the sample. In the related problem of genotype imputation, a phased reference panel is used to infer missing markers and haplotype phase of the sample (Marchini and Howie, 2010). Methods for haplotype phasing and imputation are based on computational (Halldórsson et al., 2004) and statistical inference (Browning and Browning, 2011) techniques, but both use the fact that closely spaced markers tend to be in linkage disequilibrium and smaller haplotypes blocks are often shared in a population of seemingly unrelated individuals.

In contrast, *haplotype assembly* – sometimes referred to as single individual haplotyping (Rizzi et al., 2002) – builds haplotypes for a single individual from a set of sequence reads (Schwartz, 2010). After mapping the reads on a reference genome, reads are translated into haplotype *fragments* containing only the polymorphic single nucleotide polymorphism (SNP) sites. A fragment *covers* a SNP if the corresponding sequence read contains an allele for that SNP. Because DNA sequence reads originate from a haploid chromosome, the alleles spanned by a read are assumed to exist on the same haplotype. Haplotype assembly algorithms operate on either a *SNP-fragment matrix* containing a row for each fragment and columns for SNPs or an associated graph that models the relationship between fragments or their SNP alleles.

A considerable amount of theory and algorithms have been developed for the haplotype assembly problem (Halldórsson et al., 2004, Schwartz, 2010). One approach is to restrict the input to convert an NP-hard optimization into a computationally feasible problem. For example, some authors have considered restricting the input to sequences of small read length or without mate pairs (termed gapless fragments) (He et al., 2010, Lancia et al., 2001, Rizzi et al., 2002, Bafna et al., 2005, Li et al., 2006). These models, however, are often unrealistic for current high-throughput and future third generation sequence data. Moreover, gapless fragment models are particularly problematic as paired-end sequencing is required to cover SNP alleles that are spaced at distances longer than the sequencing technology's read length. Other combinatorial and statistical algorithms have been developed for general data that relax the optimality constraint (Panconesi and Sozio, 2004, Bansal et al., 2008, DePristo et al., 2011, He et al., 2010). For example, HapCut, which was used to assemble Craig Venter's diploid genome, computes maximum cuts on a graph modeled from the fragment matrix to iteratively improve their phasing solution (Bansal and Bafna, 2008). Several of these methods were developed when Sanger was the abundant form of sequencing and thus it is unclear whether they can handle massive data on the scale of the 1000 genomes project and beyond. We will test this hypothesis for two leading haplotype assembly algorithms: the Genome Analysis ToolKit's read-backed phasing algorithm (DePristo et al., 2011) and HapCut (Bansal and Bafna, 2008). For a survey of these approaches see Schwartz, 2010.

In this paper, we present simulations to highlight the type of data needed to completely phase a chromosome and two algorithms based on local optimizations on the cycle basis of an associated graph. Our algorithms, collectively termed HapCompass, impose no prior assumptions on the input structure of the data and are generally applicable to high-throughput and third

generation sequencing technologies, including hybrid sets of reads from different technologies. HapCompass is capable of computing fast genome-wide haplotype assemblies primarily due to the low complexity of computing a spanning tree cycle basis. Futhermore, our method is faster and significantly more accurate than leading algorithms HapCut and the Genome Analysis ToolKit's (GATK) read-backed haplotype phasing algorithm using a variety of metrics on real and simulated data.

2 Methods

2.1 Graph theory models for haplotype assembly

The input to the haplotype assembly problem is a set of DNA sequence reads for a single individual. Sequence reads are mapped to the reference genome to identify the SNP content of the reads. A *fragment* is a mapped sequence read that has the non-polymorphic bases removed. SNPs for which the individual is homozygous are not useful for assembly because a fragment containing either allele cannot uniquely identify which haplotype the allele was sampled from. Furthermore, fragments containing zero or one heterozygous SNPs – while potentially useful for SNP calling – are not useful for the assembly of haplotypes and are discarded as well. Fragments containing two or more heterozygous SNPs contain valuable haplotype phase information as they link together each allele to the same haplotype and define a potential phasing.

Formally, we define the i^{th} fragment f_i as a vector of $\{0, 1, -\}$ where 0 and 1 represent the major and minor alleles at some SNP site and '-' represents a lack of information either because the read does not cover the SNP site or there was a technical failure (e.g. in mapping or sequencing). We define an $m \times n$ SNP-fragment matrix M whose m rows correspond to fragments $f_1, ..., f_m$ and n columns correspond to heterozygous SNPs $s_1, ..., s_n$. Each fragment f_i covers at least two of the n SNP sites; SNPs covered may not be consecutive on the genome, e.g. when the fragments originate from paired reads. We refer to the SNP allele k of f_i as $f_{i,k}$. We say that two fragments

f_i and f_j *conflict* if

$$\exists k | f_{i,k} \neq f_{j,k} \wedge f_{i,k} \neq \text{'}-\text{'} \wedge f_{j,k} \neq \text{'}-\text{'}$$

That is, two fragments conflict if they cover a common SNP and have different alleles at that site.

Two fundamental graph models associated to the SNP-fragment matrix M were introduced by Lancia et al., 2001 called the *fragment conflict graph* and the *SNP conflict graph.* The *fragment conflict graph*, $G_F(M) = (V_F, E_F)$, is defined as follows: the vertices are fragments, $f_i \in V_F$, $\forall i$ and the edges are $\{f_i, f_j\} \in E_F$ if f_i and f_j conflict $\forall i, j$. For an error-free M, each connected component in $G_F(M)$ has a bipartition and thus the vertices can be divided into two conflict-free disjoint subsets; the subsets define a haplotype phasing for the SNPs associated with the connected component.

The *SNP conflict graph*, $G_S(M) = (V_S, E_S)$, is defined as follows: the vertices are SNPs, $s_i \in V_S$, $\forall i$ and the edges $\{s_i, s_j\} \in E_S$ if s_i and s_j exhibit more than two haplotypes $\forall i, j$. If s_i and s_j exhibit three or four haplotypes, then some read covering s_i and s_j contains at least one error because only two haplotypes are possible for a diploid organism. Methods like HASH and HapCut employ different graph models where SNPs correspond to vertices and fragment information is encoded in the edges (Bansal et al., 2008, Bansal and Bafna, 2008). HASH and HapCut keep a reference to the current phasing of the data and each edge is weighted proportional to the number of fragments that cover the adjacent SNPs and agree with the reference phasing.

2.2 A new model: Compass graphs

Our algorithms operate on a new undirected weighted graph associated to the SNP-fragment matrix M (similar to the SNP conflict and HapCut graphs), called the *compass graph*, $G_C(M) = (V_C, E_C, w)$, defined as follows: (1) the vertices are SNPs, $s_i \in V_C$; (2) the edges are $\{s_i, s_j\} \in E_C$ if at least one fragment covers both s_i and s_j; (3) each edge $\{s_i, s_j\}$ has an associated integer weight $w(s_i, s_j)$. The weight function w is defined by the fragments. Because there exists exactly two phasings between any two heterozygous SNPs for a diploid genome, let us denote the two possible phasings as $\substack{00\\11}$ when the haplotype 00 is paired with the haplotype 11 and similarly denote $\substack{01\\10}$ the other phasing. Our weight function w for a pair of SNPs simply counts the difference between the number of $\substack{00\\11}$ phasings and the number of $\substack{01\\10}$ phasings as defined by the fragments. Formally, let F be the set of all fragments covering two SNPs s_i and s_j. The weight $w(s_i, s_j)$ is defined as follows:

$$\sum_{f_k \in F} \Big[1\Big((f_{k,i} = 1 \wedge f_{k,j} = 1) \vee (f_{k,i} = 0 \wedge f_{k,j} = 0)\Big) \\ - 1\Big((f_{k,i} = 1 \wedge f_{k,j} = 0) \vee (f_{k,i} = 1 \wedge f_{k,j} = 0)\Big)\Big]$$

where $1(b) = 1$ for b true and $1(b) = 0$ for b false. We note that a subgraph of a compass graph is also a compass graph. Figure 1 illustrates the relationship between M and its compass graph $G_C(M)$.

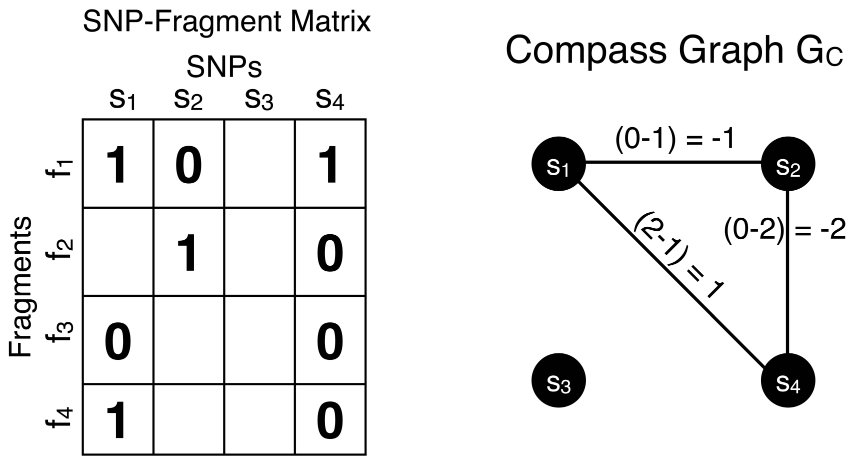


Figure 1: The SNP-fragment matrix M is shown on the left containing four fragments and four SNPs. Each pairwise phasing relationship defined by the fragments is represented on the edges of the compass graph on the right. A positive edge indicates there is more evidence in the fragments for the $^{00}_{11}$ phasing while a negative edge indicates evidence for the opposite $^{01}_{10}$ phasing.

The compass graph G_C encodes information derived from the fragment set regarding the phasings of SNPs in its edge weights. For example, fragments covering three SNPs would provide phasing information for all the $\binom{3}{2}$ edges defined by the fragment in G_C. The collected evidence for an edge may have conflicting information, that is, some fragments may provide evidence for a $^{00}_{11}$ phasing while other fragments suggest a $^{01}_{10}$ phasing. An edge with weight of zero occurs when evidence for both phasings between the pair of SNPs is equal and thus both phasings are considered. An edge with a non-zero weight is called *decisive*. A decisive edge in E_C defines the phasing between its two SNPs which is given by the sign of its weight i.e., majority rule phasing.

2.3 Problem formulations

Computing the two phased haplotypes consistent with a matrix M is trivial for error free data. When errors are present, error correction may be modeled by: removing a fragment (row), removing a SNP (column), or flipping the matrix entry defined by a particular fragment and SNP (from 0 to 1 or vice versa). Every M induces a particular G_F, G_S, and G_C, and error correction models on these graphs yield different formulations of the haplotype assembly problem. There are four problem formulations that have received the most attention in the literature:

1./2. Minimum edge/fragment removal (MER/MFR): Remove the minimum number of edges/vertices from the fragment conflict graph $G_F(M)$ such that the resulting graph is bipartite.

3. Minimum SNP removal (MSR): Remove the minimum number of vertices from the SNP conflict graph $G_S(M)$ such that no two vertices are adjacent.

4. Minimum error correction (MEC): Correct the minimum number of errors in fragments of M (by switching the allele from 0 to 1 or vice versa) such that the induced matrix M' is resolvable into two distinct haplotypes.

We note that in the MER formulation, although G_F may be completely resolvable, the resulting haplotypes may not be completely free of conflicts. A consensus SNP is commonly chosen at the construction of the haplotypes.

When the input is restricted to gapless fragments, i.e. each fragment covers a contiguous set of SNPs, MFR and MSR can be solved efficiently. However, when considering sequence reads with an arbitrary length between an arbitrary number of contiguous blocks of SNPs, MFR and MSR are NP-hard (Lancia et al., 2001). MER is NP-hard for general input (Lippert et al., 2002) and MEC is NP-hard even for gapless instances (Zhao et al., 2005, Lippert et al., 2002).

2.3.1 Minimum weighted edge removal

The *minimum weighted edge removal (MWER)* optimization problem is defined for a compass graph G_C. Let $L \subset E_C$ be a subset of edges in G_C and let G'_C be the resulting graph created from removing L from E_C. MWER aims to compute an L such that the following conditions are satisfied: (1) $\sum_{\{s_i,s_j\}\in L} |w(s_i, s_j)|$ is minimal (cost of removed edges is minimal); (2) all edges in G'_C are decisive; (3) choosing a phasing for each edge in G'_C by majority rule gives a unique phasing for G'_C. We call a subgraph of a compass graph that meets conditions (1-3) a *happy graph*.

The MWER problem for G_C aims at constructing the phased haplotypes that are most witnessed by pairwise phasing information contained in the fragments. Removed edges model the tolerance of some conflicting evidence. The final phasing for the retained edges is obtained as a consequence of the global unique phasing of the resulting happy graph.

2.4 Properties of the compass graph

We can extend unique pairwise phasings of decisive edges of G_C to unique phasings of paths. In other words, the phasing is transitive among the SNPs along a path. An edge of G_C is said to be positive (negative) if its weight is positive (negative).

Lemma 1 *There is a unique phasing between two SNPs s_i and s_j if and only if for any two simple edge-disjoint paths p and q in G_C between s_i and s_j, the number of negative edges of p plus the number of negative edges of q is even, and p and q include no 0-weight edges.*

Proof 1 *If there is a unique phasing between two SNPs s_i and s_j then they must be connected in G_C. If there is one path between s_i and s_j then the phasing is unique because this one path induces the only phasing between the two SNPs. If there is $t > 1$ paths between s_i and s_j then there exists a total of $\binom{t}{2}$ pairs of paths. Let p and q be any two paths in the traversal from s_i to s_j. We say that p and q have k and l edges with negative weight respectively. If k and l are both odd, the phasing induced between s_i and s_j by both paths is $\substack{10\\01}$. Likewise, if k and l are both even, the phasing induced between s_i and s_j by both paths is $\substack{00\\11}$. If k is odd and l is even, p defines the phasing as $\substack{10\\01}$ and q defines the phasing as $\substack{00\\11}$ (and vice versa in the case of l odd and k even). So if all paths between s_i and s_j produce a total negative edge traversal count that is even, the induced phasings cannot conflict. Likewise, if at least one pair of paths produce a total negative edge traversal count that is odd then at* *least one pair of paths disagree on the phasings of s_i and s_j. Also, if there is a unique phasing between two SNPs, no paths include a 0-weight edge by definition. The other direction follows similarly.*

Definition 1 *A compass graph is **happy** if it has a unique phasing, that is, for every pair of SNPs the phasing is unique.*

Definition 2 *A **conflicting cycle** in G_C is a simple cycle that contains an odd number of negative edges, at least one 0-weight edge or both. A non-conflicting cycle, is called a **concordant cycle** and contains an even number of negative edges and no 0-weight edges.*

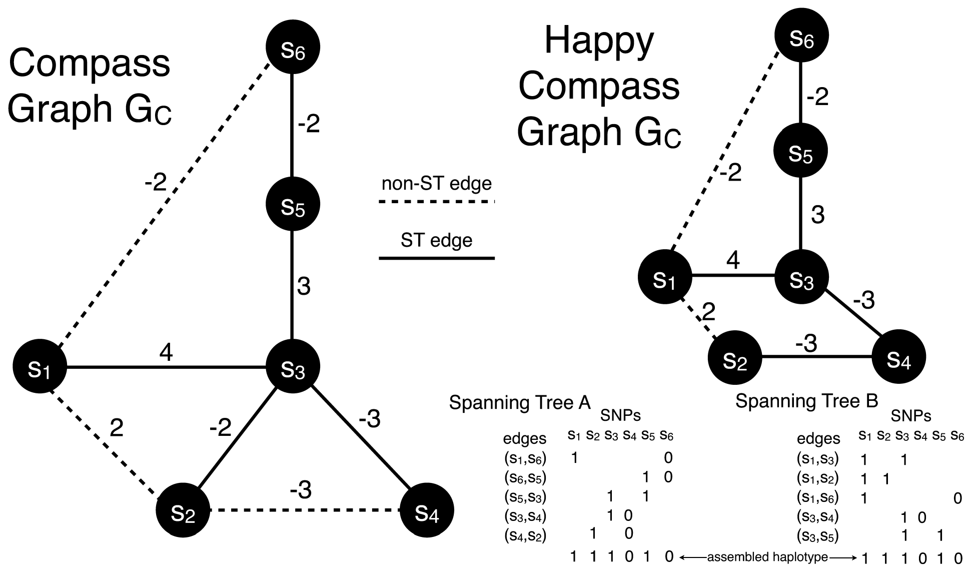
Corollary 1 *A compass graph is happy iff it has no conflicting cycles.*

In general an edge may be a member of many conflicting or concordant cycles.

A spanning tree of G_C is a connected, undirected subgraph that contains no cycles. There is a unique path between every two vertices in a spanning tree.

Theorem 1 *Every spanning tree of a compass graph is a happy graph. Every spanning tree of a happy compass graph has the same unique phasing as the compass graph.*

Figure 2 gives an example of computing a happy compass graph from G_C one edge removal step. Two spanning trees are shown in the happy G_C which correspond to the same phasing.



Spanning Tree A

edges	s_1	s_2	s_3	s_4	s_5	s_6
(s_1,s_6)	1					0
(s_6,s_5)					1	0
(s_5,s_3)			1		1	
(s_3,s_4)			1	0		
(s_4,s_2)		1		0		
	1	1	1	0	1	0

Spanning Tree B

edges	s_1	s_2	s_3	s_4	s_5	s_6
(s_1,s_3)	1		1			
(s_1,s_2)	1	1				
(s_1,s_6)	1					0
(s_3,s_4)			1	0		
(s_3,s_5)			1		1	
	1	1	1	0	1	0

SNPs

←assembled haplotype→

Figure 2: A compass graph G_C is shown on the left with two conflicting cycles. One edge removal (s_2, s_3) makes G_C happy by removing two conflicting cycles in one step. All spanning trees (ST) of the happy G_C correspond to the same phasing but only two are shown in the lower right corner.

2.5 Cycle Basis Algorithm

We present two algorithms for the minimum weighted edge removal problem on compass graphs. Our algorithms are based on optimizations involving constructing cycle bases of connected undirected weighted subgraphs of G_C. The main idea is to consider all simple cycles in an undirected graph obtained from a cycle basis. In short, we first compute a cycle basis for G_C. An efficient algorithm for generating a cycle basis first constructs a spanning tree T of G_C and defines an arbitrary root. Then, for every non-tree edge $e \in G_C$ but $e \notin T$, we form the cycle of e plus the paths from the adjacent

SNPs of e to their least common ancestor. We add the cycles created by this operation on non-tree edges to the cycle basis. This *spanning tree cycle basis* has cardinality $|E_C| - (|V_C| - 1)$.

Algorithm 1

1. Remove all 0-weight edges from G_C. *The removal of edges with 0-weight does not affect the $MWER$ score and can therefore be removed.*

2. Construct a maximum (or near maximum) spanning tree T. *A maximum weight spanning tree basis may be preferable, but computing such a basis is NP-hard (Deo et al., 1982).*

3. The spanning tree cycle basis is computed in respect to T and cycles are marked as either conflicting or concordant. Iterate (4-6) until G_C is happy:

4. Select a conflicting cycle at random and remove the edge e with weight closest to 0; this represents the edge with the least amount of evidence for phasing its SNPs. *The removal of e can either remove a tree or non-tree edge of T.*

5. If e is a non-tree edge then T is obviously still a valid spanning tree. If e is a tree edge then we add the non-tree edge e_{nt} into the spanning tree T. *After this step we clearly still have a spanning tree as any path that previously passed through the removed edge e can now pass through the added edge e_{nt}.*

6. If e was a tree edge, compute a new cycle basis in respect to $T \cup e_{nt}$. *The addition of the non-tree edge into the spanning tree T might introduce conflicts in existing concordant cycles in which case we add these cycles to the set of conflicting cycles. However, in the worst case, the algorithm will continue to remove edges until G_C is a tree which is a valid phasing thus the algorithm terminates.*

7. Output the phasing corresponding to any spanning tree of G_C. Report the number of weighted edges corrected as the score of this phasing (or report the weight of all remaining edges in G_C).

Let the $|E_C| = m$, $|V_C| = n$ and the number of non-tree edges $|E_C| - |T| = m - n + 1$.

Lemma 2 *Algorithm 1 runs in $O(m(m-n+1)^2 + (m-n+1)(m \log n))$ time.*

Proof 2 *The removal of 0-weight edges in step (1) can be done in $O(m)$ time. Step (2) involves computing a (near) maximum spanning tree which can be done in $O(m \log n)$ time. For step (3) we keep pointers at each vertex pointing to the "parent" node in respect to an arbitrary root vertex. The algorithm never traverse an edge more than $m - n + 1$ times. So this step takes no longer than $O(m(m-n+1))$. Again, step (4) takes no longer than $m(m-n+1)$ time for processing all simple cycles in respect to T. Step (5) processes one cycle, so, if the cycle being considered is c, then this operation takes at most $|c|$ time. Step (6) is dominated by $O(m \log n)$. For* *step (7) the algorithm parses through each edge of G_C thus this step takes no more than $O(m)$ time. Because we iterate through steps (4-6) at most $(m - n + 1)$ times and $m(m - n + 1) >> |c|$, the algorithmic complexity is $O(m(m - n + 1)^2 + (m - n + 1)(m \log n))$.*

Algorithm 1 is quite simple and, in practice, we use a more complex algorithm that exploits the relationship between $MWER$ and set cover.

Algorithm 2

- We follow steps (1-3) but replace (4) with a step that removes a set of highly conflicting edges. In the $MWER$ set cover formulation each edge of G_C is a set and each conflicting simple cycle is an element. The simple cycle elements belong to the edge set if the edge is part of that cycle. We then formulate the problem of resolving the conflicting cycles as finding the set of edges (sets) of minimum weight such that they cover all of the conflicting simple cycles (elements).

- Each conflicting simple cycle will have at least one edge removed, and, removing one or more edges from a conflicting cycle creates a tree which, due to Lemma 1, is non-conflicting. This, of course, would be too computationally expensive to formulate for the entire graph so we use this step on a subset of cycles. This subset is found by selecting the edge that is a member of the most conflicting cycles (this can easily be logged at the computation of the cycle basis).

- After removing a set of edges, we reconnect T. During the removal of each edge, we find the non-tree edge whose absolute value of the weight is the largest and add it back into T after all edges are removed.

- Step (6-7) is computed as before. Because the $MWER$ score is influenced by the order in which cycles are processed as well as the initial maximum spanning tree, steps (1-7) are iterated many times and the lowest score is reported as the solution.

Lemma 3 *At the end of each step, G_S is connected.*

Proof 3 *If only one cycle was corrected at a time then the non-tree edge selected for inclusion into T provides a new path for vertices previously using the removed edge. If more than one cycle was corrected by the removal of one edge, then paths previously taking the removed edge can now take any non-tree edge associated with the set of cycles.*

Lemma 3 is critically important because it ensures we do not needlessly separate components and create haplotype phase uncertainty.

The primary differences between Algorithms 1 and 2 is the local optimization step where Algorithm 2 removes multiple edges using the set cover formulation; this formulation models a sense of parsimony in that we prefer the removal of edges that resolve multiple conflicting cycles at once.

Lemma 4 *If the edge e is shared by k conflicting cycles then the removal of e resolves the k conflicting cycles.*

Proof 4 *G_C has had all edges with 0-weight removed thus each conflicting cycle has an odd number of negative edges. Let c_i and c_j be any two of the k conflicting cycles with negative edge counts of n_i and n_j. If e is positive then c_i and c_j form a cycle whose negative edge count is $n_i + n_j$. If e is negative then c_i and c_j form a cycle whose negative edge count is $(n_i - 1) + (n_j - 1)$. In both cases (odd+odd and even+even) a cycle is produced containing an even number of negative weighted cycles.*

An illustration of Lemma 4 is shown in Figure 2. There are two caveats to Lemma 4 that are due to the complex relationship between sets: (1) the removal of an edge will resolve conflicting cycles but may change concordant cycles into conflicting and (2) the removal of successive edge after the first may revert previously resolved conflicting cycles. These issues arise from the set cover formulation which simply optimizes the sum of the weighted sets and does not consider complex interactions between sets.

There are several ways to address these caveats. We may consider other properties of the edges in our minimum weighted set cover formulation. The weight on an edge e corresponds to the confidence in the pairwise phasing between the two adjacent SNPs of e. Another measure of confidence for e in G_C is the number of conflicting and concordant cycles e is a member of. The weight in the minimum weighted set cover formulation can then be computed as a combination of the edge weight and conflicting/concordant cycle membership. Because the number of conflicting or concordant cycles an edge is a member of may change with the selection of the first covering set, this minimum weighted set cover is solved iteratively. However, in practice, we specifically address (1) by breaking edge-weight ties with the number of conflicting cycles minus the number of concordant cycles and (2) by not considering shared edges from any of the resolved conflicting cycles in future removal steps of the same iteration.

Theorem 2 *Algorithm 2 is polynomial and terminates with G_C a happy graph, i.e., having exactly one phasing.*

Proof 5 *Algorithm 2 retains Algorithm 1's complexity with additional computation in step (4). The greedy approximation algorithm for set cover, however, can be computed in linear time in the size of the sets so Algorithm 2 is clearly polynomial if it terminates. Lemma 4 allows the resolution of many conflicting cycles at each local optimization step but may also change existing concordant cycles to conflicting. However, because the graph is connected at the end of each step (Lemma 3) and we correct $|E_C| - (|V_C| - 1)$ edges in the worst case, the algorithm clearly terminates. We also have the property that the final happy graph corresponds to a valid phasing because of Lemma 3.*

3 Results

The direct comparison of algorithms for which the same problem optimization is used (e.g. MEC, MFR, MSR) is straightforward. The algorithm that computes the minimum number of errors to correct is clearly the winner, for example. However, before haplotype assembly algorithms that optimize different formulations can be compared, care must be taken to develop a metric that best captures the more accurate solution.

3.1 Evaluation criteria for haplotype assembly

Before we consider new evaluation metrics that capture the quality of the haplotype assembly, we address the haplotype switch error metric that has been used previously when the ground truth is known. The haplotype switch error metric is defined as the number of switches in haplotype orientation required to reproduce the correct phasing (Lin et al., 2002). It was originally developed for the haplotype phasing problem and was among the metrics used in the Marchini et al., 2006 phasing benchmark. Switch error is generally more favorable than pure edit distances for haplotypes because it more accurately models phase relationship between adjacent SNPs.

This metric was originally developed for haplotype phasing algorithms which operate on the genotype data of many individuals simultaneously. Haplotype sharing and linkage disequilibrium are very important quantities for haplotype phasing algorithms as the relationship among adjacent SNPs allows methods to infer likely haplotypes in the data. In this manner, the switch error metric accurately captures the close range relationship between adjacent SNP phase. However, haplotype assembly algorithms operate on much different data and assumptions. Phase relationships are inferred often from long distance mate pair reads. The switch error metric does not accurately capture these relationships. Furthermore, if two haplotype assemblies do not produce the same amount of blocks of haplotypes or otherwise do not agree on where to commit to a particular phasing, then the switch error becomes biased towards those algorithms that phase less SNPs.

Instead, we suggest using a new metric inspired by genome assembly that captures how well the haplotype assembly represents the input fragments and can be applied regardless of knowing the true haplotypes. One of the most meaningful statistics for genome assembly is how many sequence reads successfully map back to the assembly. We can also slightly modify this metric to ask the question: "How many of the phase relationships represented by the read fragments are represented in the assembly?" This *fragment mapping phase relationship* (FMPR) metric summarizes how well the haplotype assembly represents the input data.

Let the set of all fragments be F and f_i the i^{th} fragment of F. We denote the k^{th} SNP of f_i as $f_{i,k}$. The haplotypes produced from an algorithm are denoted h_1 and h_2 and the allele of h_1 at position k is denoted $h_{1,k}$ ($h_{2,k}$ is defined similarly). Then the fragment mapping phase relationship metric can be described as

$$\sum_{f_i \in F} \sum_{f_{i,j}, f_{i,k} \in f_i | j \neq k} min\,(1(f_{i,j}, f_{i,k}, h_1), 1(f_{i,j}, f_{i,k}, h_2))$$

where 1() is a function that takes two SNP alleles and a haplotype and determines whether the phase relationship between the two alleles exists in the haplotype; formally, $1(f_{i,j}, f_{i,k}, h_1) = 1$ if $(f_{i,j} \neq \text{'}-\text{'} \wedge f_{i,k} \neq \text{'}-\text{'}) \wedge (f_{i,j} \neq h_{1,j} \wedge f_{i,k} \neq h_{1,k})$ and $1(f_{i,j}, f_{i,k}, h_1) = 0$ otherwise. This metric is computed by counting all of the pairwise phase relationships defined by the input set of fragments that do not exist in the solution. One fortunate side effect of this metric is that an algorithm that produces smaller blocks will be penalized. For instance, if an algorithm produces a haplotype assembly for five disjoint blocks when fragments exist in the data that connect every SNP in one large block, the switch error metric will not penalize the unknown phase between blocks. However, the fragment mapping metric will capture the phase ambiguity error that exists between disjoint blocks if the input fragments do indeed suggest they should be connected. We also define the boolean fragment mapping (BFM) metric which counts the percentage of fragments that map to the resolved haplotypes with at least one error. For both FMPR and BFM metrics, smaller numbers are preferred.

3.2 1000 Genomes data

We first evaluate the number of reads that must supplement the current high coverage 1000 genomes data (The 1000 Genomes Project Consortium, 2010) for the NA12878 CEU individual in order to achieve a complete haplotype assembly of chromosome 22. To do this, we supplemented the 454, Illumina, and SOLiD sequence data with simulated Illumina reads. The starting point of each simulated read was generated at random from the set of bases that were sampled by real sequence reads. Illumina-sized reads were simulated using varying distributions for insert size. Figure 3 shows a least squares fitted curve to the largest component (or block) sizes for various coverages in chromosome 22. Disconnected components of G_C must be phased separately and the haplotype phase between them is ambiguous; therefore, the largest component size gives an indication of the connectedness of G_C and the size of the maximum achievable phased haplotype.

We then evaluated each algorithm using the aforementioned metrics for the Illumina, SOLiD, and 454 reads generated for the CEU individual NA12878 in the 1000 genomes data (Table 1). Because each sequencing technology produces reads with similar insert sizes, the real data block sizes are small. For these block sizes, HapCompass produces the best results with GATK also producing very accurate haplotype assemblies.

3.3 Simulated data

Limitations in current sequencing technologies restrict the number of SNPs one can hope to phase from the sequence reads. Many factors influence the connectedness of G_C but the most influential factor is the mean sizes and variance of the inserts used to generate the paired reads (Halldorsson et al., 2011). This is less of a concern for whole-exome data where haplotype assemblies can be constructed rather easily with high coverage. However, in order to test the algorithms on their capability to provide genome-wide haplotype assemblies in terms of both accuracy and time efficiency, we simulated two datasets of 10 million 100 bp reads and varied the error parameter. The 10

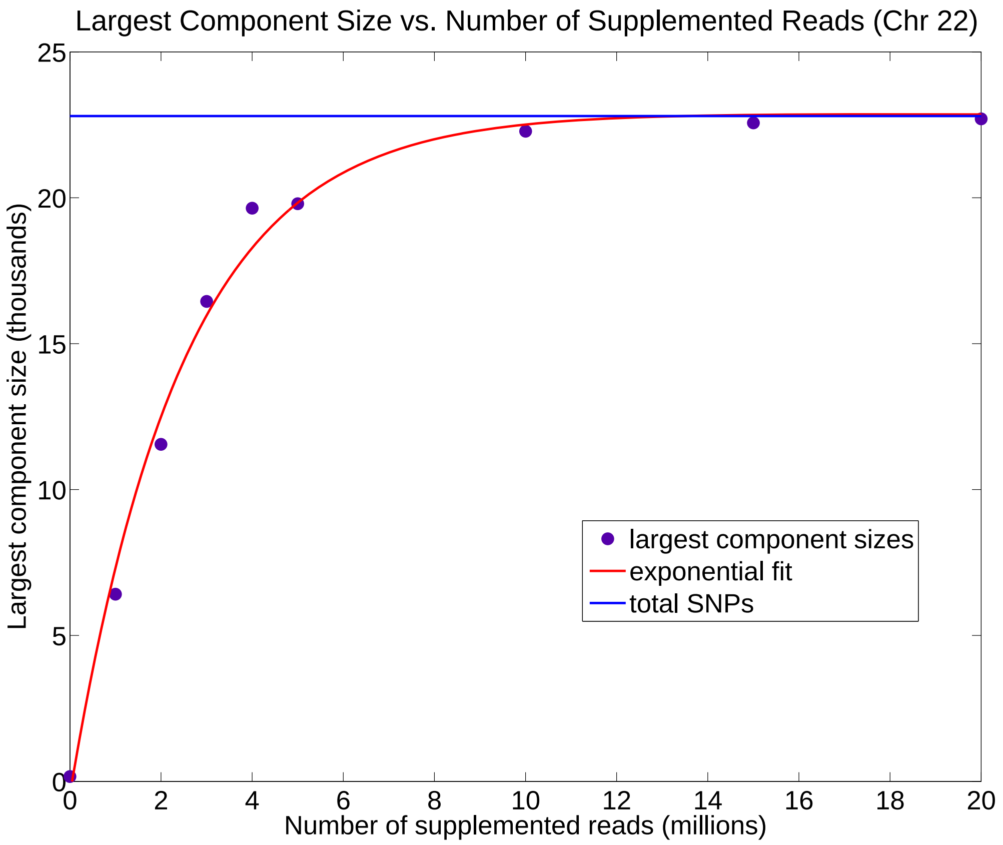


Figure 3: In this simulation study, reads of size 100bp were simulated on chromosome 22 of the 1000 genomes CEU individual NA12878. Mate pair lengths were sampled at random from one of four normal distribution with means [10kb, 50kb, 100kb, 250kb] and standard deviations [1kb, 5kb, 10kb, 25kb]. With these parameters for sequencing, we require about 10 million reads to connect most of the SNPs of G_C.

block size	no. fragments	HapCut FMPR (BFM %)	GATK FMPR (BFM %)	HC FMPR (BFM %)
51	477	223 (13.8)	60 (8.4)	**23 (1.9)**
53	581	265 (11)	30 (2.9)	**25 (2.4)**
53	551	71 (7.1)	23 (2.9)	**9 (1.3)**
58	626	209 (11)	**12 (1.4)**	**12 (1.4)**
60	645	199 (10.1)	54 (3.9)	**43 (3.1)**
60	467	28 (4.7)	18 (3)	**4 (0.86)**
62	393	24 (4.5)	14 (3.1)	**6 (1.5)**
62	528	126 (10.6)	16 (2.5)	**8 (1.3)**
63	770	45 (3.8)	24 (2.2)	**19 (1.7)**
66	602	91 (5.6)	31 (3.7)	**11 (1.5)**
66	718	452 (14.6)	47 (3.3)	**28 (2.1)**
79	877	245 (10.1)	26 (2.1)	**8 (0.8)**
102	949	212 (8.7)	48 (2.7)	**37 (1.9)**
166	1914	207 (5.9)	83 (2.7)	**44 (1.5)**
Total FMPR	-	2397	486	277

Table 1: HapCut, GATK, and HapCompass (HC) were evaluated according to the fragment mapping phase relationship and boolean fragment mapping metrics for 1000 genomes data chromosome 22 of individual NA12878. The *block size* is the number of SNPs in the component of G_C and *no. fragments* denotes how many read fragments were used for assembly. Bold cells denote the algorithm with the best score.

block size	no. fragments	HapCut FMPR (BFM %)	GATK FMPR (BFM %)	HC FMPR (BFM %)
580	2268	355 (13.8)	703 (28.2)	**284 (11.4)**
1331	4023	647 (14.5)	1236 (28.7)	**441 (10.1)**
1598	6545	1182 (15.5)	2011 (27.6)	**1033 (13.9)**
1835	6962	1212 (14.8)	2235 (28.8)	**1089 (13.8)**
3193	15036	3416 (17.7)	5237 (30)	**2746 (15.7)**
4153	17862	3642 (16.6)	- (-)	**2719 (13.2)**

Table 2: HapCut, GATK, and HapCompass (HC) were evaluated according to the fragment mapping phase relationship and boolean fragment mapping metrics for 1000 genomes data chromosome 22 of individual NA12878 and 10 million simulated reads with error rate = 0.05 and read length = 100. A dash (-) mark denotes the algorithm did not finish using the allotted resources. Bold cells denote the algorithm with the best score.

million reads parameter is guided by the data generated for Figure 3 and will vary depending on read length, insert size distributions, coverage and genome allele structure (e.g. runs of homozygosity that are longer than the insert size will disconnect components of G_C).

Because the NA12878 individual is the child of a CEU trio who were also sequenced, we used the parents to phase most of the SNPs; a random phasing was selected for SNPs that were triply heterozygous. Using this method, we are able to construct a set of haplotypes to simulated reads from that are as close to the ground truth as possible with the available data. Our principle measurements of accuracy are FMPR and BFM. First we tested each algorithm on simulated data with moderately high error rates (0.05).

We can summarize the trends in Tables 1 and 2 by fitting a linear least squares regression line to the data (Figure 4).

It is clear from Figure 4 that HapCompass produces the best results. HapCut seems to produce better results than GATK on larger haplotype blocks (the reverse was true for the small haplotype blocks from real data). When considering serial execution, the processing times for HapCut and HapCompass were similar. For instance, for the simulated component of size 4177, 25 iterations of HapCut took 3.7 hours while 25 iterations of HapCompass took 4.8 hours. However, iterations of the HapCompass algorithm are independent and can be trivially parallelized. When this is the case, the solution with the smallest $MWER$ score is retained as the overall solution. HapCut and HapCompass both used less than 2 gigabytes of memory while GATK required a great deal more memory and processing time for similar sized components. Each algorithm was terminated if it required more than 12 hours of processing time or 8 gigabytes of heap space.

Even though switch error has an unclear interpretation on haplotype assembly data, we show that switch error produces the same algorithmic rankings. Switch error is as defined before but we incur a penalty of 1 for each haplotype block reported beyond the first. Because we compare each algorithm to a connected haplotype block – in the sense that there is a path between every SNP in G_C – reporting more than one phasing represents a switch error between phased components. The switch error metric gives the same relative ranking of algorithm performance (Table 3). We only show these results for completeness and do not recommend using switch error as

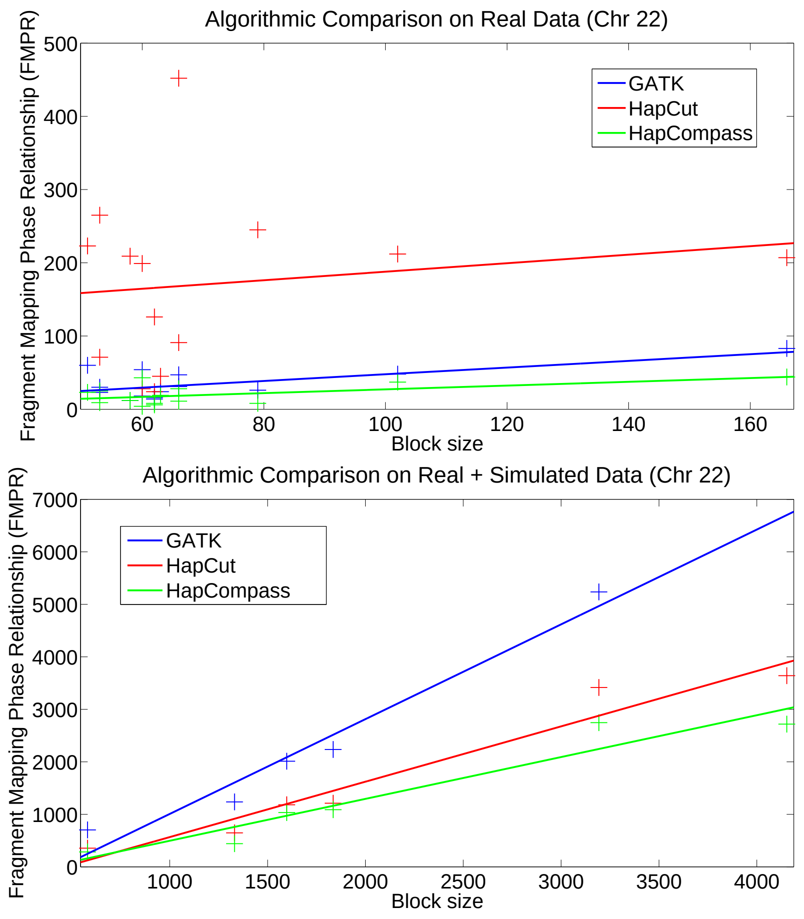


Figure 4: We fit a linear least squares regression line to the FMPR measurement for algorithms Genome Analysis ToolKit (GATK), HapCut, and HapCompass on chromosome 22 of the (Top) 1000 genomes data for the NA12878 individual (Table 1) and (Bottom) 1000 genomes data for the NA12878 individual with 10 million simulated reads of length 100 with sequence base error rate of 0.05 (Table 2).

block size	no. fragments	HapCut SE	GATK SE	HC SE
580	2268	197	259	**148**
1331	4023	544	581	**329**
1598	6545	604	654	**474**
1835	6962	694	766	**563**
3193	15036	1092	1287	**859**
4153	17862	1630	-	**1007**

Table 3: Switch error (SE) measurements for HapCut, GATK, and HapCompass (HC) for the same data as Table 2. A dash (-) mark denotes the algorithm did not finish using the allotted resources. Bold cells denote the algorithm with the best score.

the sole measurement of haplotype assembly algorithm accuracy.

We then reduced the error rate to evaluate the behavior of each algorithm on higher quality data. Table 4 again demonstrates that HapCompass remains significantly better than HapCut and GATK.

4 Discussion

The HapCompass algorithmic framework can also be extended to other optimization formulations. For instance, it is possible to extend these algorithms to accommodate the MEC formulation by employing a different local optimization step. However, we do not have the same termination requirements as with MWER. In particular, an edge must have all fragment based evidence suggesting one unique phasing. The MEC optimization also defines a different local optimization sub-problem. Preliminary results are encourag-

block size	no. fragments	HapCut FMPR (BFM %)	GATK FMPR (BFM %)	HC FMPR (BFM %)
578	2326	180 (6.6)	650 (25.5)	**46 (1.7)**
1852	7234	888 (10.7)	1896 (23.7)	**207 (2.5)**
4177	17953	2088 (9.3)	- (-)	**425 (2)**

Table 4: HapCut, GATK, and HapCompass (HC) were evaluated according to the fragment mapping phase relationship and boolean fragment mapping metrics for 1000 genomes data chromosome 22 of individual NA12878 and 10 million simulated reads with error rate = 0.01 and read length = 100. A dash (-) mark denotes the algorithm did not finish using the allotted resources. Bold cells denote the algorithm with the best score.

ing where we model the local optimization as a haplotype clustering problem; the sub-problem becomes finding the two haplotypes that minimize the MEC on a subgraph of G_C.

As is the case with most haplotype assembly algorithms, our algorithm does not yet consider quality scores of sequence reads or SNP calls. In the previous analyses we simulated reads with perfect mapping and sequence call quality. This information, however, could be incorporated into the weights on the edges. The edges of G_C require a measurement of confidence in the phasing and other sources of information may be encoded on these edges; for instance, a likelihood model may also be formulated accommodating the inclusion of sequence base call scores and the quality of SNP calls.

5 Conclusion

Haplotype assembly is becoming increasingly important as the cost of sequencing plummets and more genome-wide and whole-exome studies are conducted (Levy et al., 2007, Tewhey et al., 2011). We have designed and implemented a haplotype assembly algorithm that is widely applicable to these studies because it does not make any prior assumptions on the input data. Through the use of simulations, we show that supplementing 1000 genomes data with sequencing data of a particular type connects G_C enabling the haplotype assembly of entire chromosomes. We described the fragment mapping phase relationship and boolean fragment mapping metrics that capture the quality of the haplotype assembly through support from mapped fragments. These metrics can be used independent of the algorithm and without knowing the true haplotypes to evaluate the quality of the haplotype assembly.

We compared HapCompass to two leading haplotype assembly software packages that can also process arbitrary input sequence data: HapCut and the Genome Analysis ToolKit's read-backed phasing algorithm. HapCompass is shown to be more accurate on real 1000 genomes data for the BFM and FMPR metrics. We also show that HapCompass is more accurate when we supplement the existing 1000 genomes real data with simulated Illumina reads for BFM, FMPR and haplotype switch metrics on haplotype blocks of unprecedented size. Although only data from chromosome 22 is shown (chr22 with 33144 SNPs), the number of SNPs on the most polymorphic chromosome (chr2 with 218005 SNPs) is only an order of magnitude more according to CEU 1000 genomes data (minor allele frequency ≥ 0.01). Because we have shown the small order polynomial complexity of this algorithm, we do not believe it would be difficult to extend this algorithm to operate on large components of G_C in parallel on a computing cluster for genome-wide data. As high-throughput sequencing becomes more available to a greater number of researchers, we believe HapCompass will provide a valuable tool to quickly and accurately identify the haplotypes of diploid organisms.

Acknowledgments

We thank Bjarni Halldórsson for many helpful early discussions regarding haplotype assembly. This work was supported by National Science Foundation grant number 1048831 to S.I. The work of D.A. and S.I. was supported by this grant.

Author Disclosure Statement.

No competing financial interests exist.

References

Vineet Bafna, Sorin Istrail, Giuseppe Lancia, and Romeo Rizzi. Polynomial and apx-hard cases of the individual haplotyping problem. *Theoretical*

Computer Science, 335(1):109 – 125, 2005. ISSN 0304-3975. doi: 10.1016/j.tcs.2004.12.017. URL `http://www.sciencedirect.com/science/article/pii/S0304397504008114`. ¡ce:title¿Pattern Discovery in the Post Genome¡/ce:title¿.

Vikas Bansal and Vineet Bafna. Hapcut: an efficient and accurate algorithm for the haplotype assembly problem. *Bioinformatics*, 24(16):153–159, 2008.

Vikas Bansal, Aaron L. Halpern, Nelson Axelrod, and Vineet Bafna. An mcmc algorithm for haplotype assembly from whole-genome sequence data. *Genome Research*, 18(8):1336–1346, 2008. doi: 10.1101/gr.077065.108. URL `http://genome.cshlp.org/content/18/8/1336.abstract`.

Sharon R. Browning and Brian L. Browning. Haplotype phasing: existing methods and new developments. *Nat Rev Genet*, 12(10):703–714, October 2011. ISSN 1471-0064. doi: 10.1038/nrg3054. URL `http://dx.doi.org/10.1038/nrg3054`.

Narsingh Deo, G. Prabhu, and M. S. Krishnamoorthy. Algorithms for Generating Fundamental Cycles in a Graph. *ACM Trans. Math. Softw.*, 8 (1):26–42, 1982. ISSN 0098-3500. doi: 10.1145/355984.355988. URL `http://dx.doi.org/10.1145/355984.355988`.

Mark A. DePristo, Eric Banks, Ryan Poplin, Kiran V. Garimella, Jared R. Maguire, Christopher Hartl, Anthony A. Philippakis, Guillermo del Angel, Manuel A. Rivas, Matt Hanna, Aaron McKenna, Tim J. Fennell, Andrew M. Kernytsky, Andrey Y. Sivachenko, Kristian Cibulskis, Stacey B. Gabriel, David Altshuler, and Mark J. Daly. A framework for variation discovery and genotyping using next-generation DNA sequencing data. *Nat Genet*, 43(5):491–498, May 2011. ISSN 1061-4036. doi: 10.1038/ng.806. URL `http://dx.doi.org/10.1038/ng.806`.

Bjarni V. Halldórsson, Vineet Bafna, Nathan Edwards, Shibu Yooseph, and Sorin Istrail. A survey of computational methods for determining haplotypes. In *Computational Methods for SNPs and Haplotype Inference (LNCS 2983)*, pages 26–47. Springer, 2004. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.85.4196`.

Bjarni V. Halldorsson, Derek Aguiar, and Sorin Istrail. Haplotype phasing by multi-assembly of shared haplotypes: Phase-dependent interactions between rare variants. In *Proceedings of the Pacific Symposium on Biocomputing*, pages 88–99, 2011.

Dan He, Arthur Choi, Knot Pipatsrisawat, Adnan Darwiche, and Eleazar Eskin. Optimal algorithms for haplotype assembly from whole-genome sequence data. *Bioinformatics*, 26(12):i183–i190, 2010. doi: 10.1093/bioinformatics/btq215. URL `http://bioinformatics.oxfordjournals.org/content/26/12/i183.abstract`.

Peter M. Krawitz, Michal R. Schweiger, Christian Rodelsperger, Carlo Marcelis, Uwe Kolsch, Christian Meisel, Friederike Stephani, Taroh Kinoshita, Yoshiko Murakami, Sebastian Bauer, Melanie Isau, Axel Fischer, Andreas Dahl, Martin Kerick, Jochen Hecht, Sebastian Kohler, Marten Jager, Johannes Grunhagen, Birgit J. de Condor, Sandra Doelken, Han G. Brunner, Peter Meinecke, Eberhard Passarge, Miles D. Thompson, David E. Cole, Denise Horn, Tony Roscioli, Stefan Mundlos, and Peter N. Robinson. Identity-by-descent filtering of exome sequence data identifies PIGV mutations in hyperphosphatasia mental retardation syndrome. *Nature Genetics*, 42(10):827–829, August 2010. ISSN 1061-4036. doi: 10.1038/ng.653. URL `http://dx.doi.org/10.1038/ng.653`.

Giuseppe Lancia, Vineet Bafna, Sorin Istrail, Ross Lippert, and Russell Schwartz. SNPs Problems, Complexity, and Algorithms. In *ESA '01: Proceedings of the 9th Annual European Symposium on Algorithms*, pages 182–193, London, UK, 2001. Springer-Verlag. ISBN 3-540-42493-8. URL `http://portal.acm.org/citation.cfm?id=740484`.

Samuel Levy, Granger Sutton, Pauline C. Ng, Lars Feuk, Aaron L. Halpern, Brian P. Walenz, Nelson Axelrod, Jiaqi Huang, Ewen F. Kirkness, Gennady Denisov, Yuan Lin, Jeffrey R. MacDonald, Andy Wing Chun W. Pang, Mary Shago, Timothy B. Stockwell, Alexia Tsiamouri, Vineet Bafna, Vikas Bansal, Saul A. Kravitz, Dana A. Busam, Karen Y. Beeson, Tina C. McIntosh, Karin A. Remington, Josep F. Abril, John Gill, Jon Borman, Yu-Hui H. Rogers, Marvin E. Frazier, Stephen W. Scherer, Robert L. Strausberg, and J. Craig Venter. The diploid genome sequence of an individual human. *PLoS biology*, 5(10):e254+, September 2007. ISSN 1545-7885. doi: 10.1371/journal.pbio.0050254. URL `http://dx.doi.org/10.1371/journal.pbio.0050254`.

Zhen-ping Li, Ling-yun Wu, Yu-ying Zhao, and Xiang-sun Zhang. A dynamic programming algorithm for the k-haplotyping problem. *Acta Mathematicae Applicatae Sinica (English Series)*, 22:405–412, 2006. ISSN 0168-9673. URL `http://dx.doi.org/10.1007/s10255-006-0315-6`. 10.1007/s10255-006-0315-6.

Shin Lin, David J. Cutler, Michael E. Zwick, and Aravinda Chakravarti. Haplotype inference in random population samples. *The American Journal of Human Genetics*, 71(5):1129–1137, Nov 2002.

R. Lippert, R. Schwartz, G. Lancia, and S. Istrail. Algorithmic strategies for the single nucleotide polymorphism haplotype assembly problem. *Brief Bioinform*, 3(1):23–31, March 2002. ISSN 1467-5463. doi: 10.1093/bib/3.1.23. URL `http://dx.doi.org/10.1093/bib/3.1.23`.

J. Marchini, D. Cutler, N. Patterson, M. Stephens, E. Eskin, E. Halperin, S. Lin, Z. S. Qin, H. M. Munro, G. R. Abecasis, and P. Donnelly. A comparison of phasing algorithms for trios and unrelated individuals. *American Journal of Human Genetics*, 78(3):437–450, March 2006. ISSN 0002-9297. doi: 10.1086/500808. URL `http://dx.doi.org/10.1086/500808`.

Jonathan Marchini and Bryan Howie. Genotype imputation for genome-wide association studies. *Nat Rev Genet*, 11(7):499–511, June 2010. ISSN 1471-0056. doi: 10.1038/nrg2796. URL `http://dx.doi.org/10.1038/nrg2796`.

Alessandro Panconesi and Mauro Sozio. Fast hare: A fast heuristic for single individual snp haplotype reconstruction. In *Proceedings of the 4th International Workshop on Algorithms in Bioinformatics WABI04*, volume 3240, pages 266 – 277, 2004.

Tyler Mark Pierson, Dimitre R. Simeonov, Murat Sincan, David A. Adams, Thomas Markello, Gretchen Golas, Karin Fuentes-Fajardo, Nancy F. Hansen, Praveen F. Cherukuri, Pedro Cruz, Craig Blackstone, Cynthia Tifft, Cornelius F. Boerkoel, and William A. Gahl. Exome sequencing and snp analysis detect novel compound heterozygosity in fatty acid hydroxylase-associated neurodegeneration. *Eur J Hum Genet*, 20(4):476–479, 2012.

Romeo Rizzi, Vineet Bafna, Sorin Istrail, and Giuseppe Lancia. Practical Algorithms and Fixed-Parameter Tractability for the Single Individual SNP Haplotyping Problem. In *Algorithms in Bioinformatics*, volume 2452 of *Lecture Notes in Computer Science*, pages 29–43. Springer Berlin Heidelberg, 2002. ISBN 978-3-540-44211-0.

Russell Schwartz. Theory and algorithms for the haplotype assembly problem. *Commun. Inf. Syst.*, 10(1):23–38, 2010.

Ryan Tarpine, Fumei Lam, and Sorin Istrail. Conservative extensions of linkage disequilibrium measures from pairwise to multi-loci and algorithms for optimal tagging SNP selection. In *Proceedings of the 15th Annual International Conference on Research in Computational Molecular Biology*, RECOMB'11, pages 468–482, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-20035-9. URL `http://dl.acm.org/citation.cfm?id=1987587.1987629`.

Ryan Tewhey, Vikas Bansal, Ali Torkamani, Eric J. Topol, and Nicholas J. Schork. The importance of phase information for human genomics. *Nature Reviews Genetics*, 12(3):215–223, February 2011. ISSN 1471-0056. doi: 10.1038/nrg2950. URL `http://dx.doi.org/10.1038/nrg2950`.

The 1000 Genomes Project Consortium. A map of human genome variation from population-scale sequencing. *Nature*, 467(7319):1061–1073, October 2010. ISSN 0028-0836. doi: 10.1038/nature09534. URL `http://dx.doi.org/10.1038/nature09534`.

Yu-Ying Zhao, Ling-Yun Wu, Ji-Hong Zhang, Rui-Sheng Wang, and Xiang-Sun Zhang. Haplotype assembly from aligned weighted SNP fragments. *Computational Biology and Chemistry*, 29(4):281 – 287, 2005. ISSN 1476-9271. doi: 10.1016/j.compbiolchem.2005.05.001. URL `http://www.sciencedirect.com/science/article/pii/S1476927105000459`.